

# A New Fully Polynomial Time Approximation Scheme for the Interval Subset Sum Problem

Rui Diao · Ya-Feng Liu · Yu-Hong Dai

Received: date / Accepted: date

**Abstract** The interval subset sum problem (ISSP) is a generalization of the well-known subset sum problem. Given a set of intervals  $\{[a_{i,1}, a_{i,2}]\}_{i=1}^n$  and a target integer  $T$ , the ISSP is to find a set of integers, at most one from each interval, such that their sum best approximates the target  $T$  but cannot exceed it. In this paper, we first study the computational complexity of the ISSP. We show that the ISSP is relatively easy to solve compared to the 0-1 Knapsack problem (KP). We also identify several subclasses of the ISSP which are polynomial time solvable (with high probability), albeit the problem is generally NP-hard. Then, we propose a new fully polynomial time approximation scheme (FPTAS) for solving the general ISSP problem. The time and space complexities of the proposed scheme are  $\mathcal{O}(n \max\{1/\epsilon, \log n\})$  and  $\mathcal{O}(n + 1/\epsilon)$ , respectively, where  $\epsilon$  is the relative approximation error. To the best of our knowledge, the proposed scheme has almost the same time complexity but a significantly lower space complexity compared to the best known scheme. Both the correctness and efficiency of the proposed scheme are validated by numerical simulations. In particular, the proposed scheme successfully solves ISSP instances with  $n = 100,000$  and  $\epsilon = 0.1\%$  within one second.

**Keywords** interval subset sum problem · computational complexity · solution structure · fully polynomial time approximation scheme · worst-case performance

**Mathematics Subject Classification (2010)** 90C59 · 68Q25

---

Y.-F. Liu was partially supported by NSFC Grants 11671419, 11331012, 11631013, and 11571221. Y.-H. Dai was partially supported by the Key Project of Chinese National Programs for Fundamental Research and Development Grant 2015CB856000, and NSFC Grants 11631013, 11331012, and 71331001.

---

R. Diao, Y.-F. Liu (Corresponding author), and Y.-H. Dai  
State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China  
E-mail: diaorui@lsec.cc.ac.cn; yafliu@lsec.cc.ac.cn; dyh@lsec.cc.ac.cn

## 1 Introduction

The subset sum problem (SSP) is a fundamental problem in complexity theory and cryptography. The optimization formulation of the SSP is given as follows:

$$\begin{aligned} \max_x \quad & \sum_{i=1}^n a_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^n a_i x_i \leq T, \\ & x_i \in \{0, 1\}, \quad i = 1, 2, \dots, n, \end{aligned} \tag{1.1}$$

where  $\{a_i\}_{i=1}^n$  and  $T$  are some given positive integers.

The SSP is a famous NP-hard problem [12]. Therefore, all exact algorithms for the SSP are not polynomial unless  $P=NP$ . The classical pseudo-polynomial<sup>1</sup> algorithm based on the dynamic programming technique for solving the SSP has  $\mathcal{O}(nT)$  time and space complexities. An algorithm with an improved complexity  $\mathcal{O}(n \max_{1 \leq i \leq n} a_i)$  was proposed by Pisinger [13]. Various fully polynomial time approximation schemes (FPTAS<sup>2</sup>) have also been proposed for the SSP. The first FPTAS for the SSP was proposed by Ibarra and Kim [5] in 1975, which has a time complexity of being  $\mathcal{O}(n/\epsilon^2)$  and a space complexity of being  $\mathcal{O}(n + 1/\epsilon^3)$ . To the best of our knowledge, the current best FPTAS for the SSP was proposed by Kellerer and Pferschy [6]. The time complexity and space complexity of the proposed FPTAS in [6] are  $\mathcal{O}(\min\{n/\epsilon, n + 1/\epsilon^2 \log(1/\epsilon)\})$  and  $\mathcal{O}(n + 1/\epsilon)$ , respectively. There are also some works focusing on characterizing the easy subclass of the SSP. For instance, in [1, 4, 9], both theory and algorithms were proposed for the SSP when  $n/\log_2 \max_{1 \leq i \leq n} a_i$  is small.

The 0-1 Knapsack problem (KP) is a generalization of the SSP, which has the optimization form as follows:

$$\begin{aligned} \max_x \quad & \sum_{i=1}^n v_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^n a_i x_i \leq T, \\ & x_i \in \{0, 1\}, \quad i = 1, 2, \dots, n, \end{aligned} \tag{1.2}$$

where  $\{v_i\}_{i=1}^n$ ,  $\{a_i\}_{i=1}^n$ , and  $T$  are some given positive integers. When  $v_i = a_i$  for all  $i = 1, 2, \dots, n$ , the 0-1 KP reduces to the SSP.

<sup>1</sup> An algorithm that solves a problem is called a pseudo-polynomial time algorithm if its time complexity function is bounded above by a polynomial function related to the numeric value of the input, but exponential in the length of the input.

<sup>2</sup> An algorithm is called an FPTAS for a maximization problem if, for any given instance of the problem and any relative error  $\epsilon \in (0, 1)$ , the algorithm returns a solution value  $v^A$  satisfying  $v^A \geq (1 - \epsilon)v^*$ , where  $v^*$  is the optimal value of the corresponding instance, and its time complexity function is polynomial both in the length of the given data of the problem and in  $1/\epsilon$ .

Various FPTASs have also been proposed for the 0-1 KP. For instance, Lawler [10] proposed an FPTAS for the 0-1 KP with time and space complexities being  $\mathcal{O}(n \log(1/\epsilon) + 1/\epsilon^4)$  and  $\mathcal{O}(n + 1/\epsilon^3)$ , respectively. Later, Magazine and Oguz [11] proposed another PFTAS for the 0-1 KP. The time and space complexities of the proposed FPTAS in [11] are  $\mathcal{O}(n^2 \log n/\epsilon)$  and  $\mathcal{O}(n/\epsilon)$ , respectively. A relatively recent FPTAS, with the time complexity  $\mathcal{O}(n \log n + \min\{n, 1/\epsilon \log(1/\epsilon)\}1/\epsilon^2)$  and the space complexity  $\mathcal{O}(n + 1/\epsilon^2)$ , was proposed by Kellerer and Pferschy [7]. These FPTASs for the 0-1 KP are summarized in Table 1.1.

Another generalization of the SSP is the interval subset sum problem (ISSP), which is the focus of this paper. Mathematically, the ISSP can be formulated as follows:

$$\begin{aligned} \max_x \quad & \sum_{i=1}^n x_i \\ \text{s.t.} \quad & \sum_{i=1}^n x_i \leq T, \\ & x_i \in \{0\} \cup [a_{i,1}, a_{i,2}], \quad x_i \in \mathbb{Z}, \quad i = 1, 2, \dots, n, \end{aligned} \tag{1.3}$$

where  $a_{i,2} \geq a_{i,1}$ ,  $i = 1, 2, \dots, n$  are positive integers and  $\mathbb{Z}$  denotes the set of integers. When  $a_{i,1} = a_{i,2}$  for all  $i = 1, 2, \dots, n$ , the ISSP reduces to the SSP.

The ISSP was first studied by Kothari *et al.* [8] with applications in auction clearing for uniform-price multi-unit auctions. The ISSP has also found wide applications in unit commitment, power generation [2], and many others. For instance, in dispatch of the power system, the electric power units need be operated to match the total power load. Each power unit can be chosen to be off or on and the output of each power unit can be adjusted in an interval when it is on. These features can be represented and formulated as the constraints in problem (1.3). Currently, the FPTAS proposed by Kothari *et al.* [8] is the only known algorithm designed for solving the ISSP. Both the time complexity and the space complexity of the FPTAS in [8] are  $\mathcal{O}(n/\epsilon)$ .

In this paper, we consider the ISSP and propose a new efficient FPTAS for solving it. Compared to the FPTAS for the ISSP in [8], the proposed FPTAS in this paper has almost the same time complexity but a significantly lower space complexity. Some of the existing FPTASs for the SSP, the ISSP and the 0-1 KP are summarized in Table 1.1. The main contributions of this paper are listed as follows.

- The ISSP is shown to be easier than the 0-1 KP in the sense that the ISSP can be equivalently reformulated as a 0-1 KP and therefore any algorithms for the 0-1 KP can be applied to solve the ISSP (see Theorem 2.1);
- Some polynomial time solvable subclasses of the ISSP are identified (see Theorems 2.2 and 2.3). For instance, it is shown in Theorem 2.3 that the ISSP is polynomial time solvable when  $a_{i,2} \geq 2a_{i,1}$  for all  $i = 1, 2, \dots, n$ ;
- By exploiting a new solution structure of the ISSP (see Lemma 3.4), a new FPTAS is proposed to solve the ISSP. The proposed FPTAS enjoys a significantly lower space complexity compared to the existing one; See Theorems 3.10 and 3.11.
- Both the correctness and efficiency of the proposed FPTAS are shown by applying it to solve large scale ISSP instances. In particular, the proposed FPTAS

is capable of solving ISSP instances with  $n = 100,000$  and  $\epsilon = 0.1\%$  within one second; see Section 5.

The rest of this paper is organized as follows. In Section 2, we study the computational complexity of the ISSP. In Section 3, we propose a new FPTAS for the ISSP and analyze the time and space complexities of the proposed scheme. Simulation results are presented in Section 4 to validate the correctness and efficiency of the proposed FPTAS. The C++ simulation codes are available at [<http://bitbucket.org/diaorui/issp>]. Finally, some concluding remarks are drawn in Section 5.

To streamline the presentation, all proofs of Lemmas/Theorems/Corollaries in this paper are relegated to Appendix A.

**Table 1.1** Summary of FPTASs for SSP, ISSP, and 0-1 KP

Problem	Time Complexity	Space Complexity	Reference
SSP	$\mathcal{O}(\min\{n/\epsilon, n + 1/\epsilon^2 \log(1/\epsilon)\})$	$\mathcal{O}(n + 1/\epsilon)$	[6]
ISSP	$\mathcal{O}(n/\epsilon)$	$\mathcal{O}(n/\epsilon)$	[8]
	$\mathcal{O}(n \max\{1/\epsilon, \log n\})$	$\mathcal{O}(n + 1/\epsilon)$	this paper
0-1 KP	$\mathcal{O}(n \log(1/\epsilon) + 1/\epsilon^4)$	$\mathcal{O}(n + 1/\epsilon^3)$	[10]
	$\mathcal{O}(n^2 \log n/\epsilon)$	$\mathcal{O}(n/\epsilon)$	[11]
	$\mathcal{O}(n \log n + \min\{n, 1/\epsilon \log(1/\epsilon)\}1/\epsilon^2)$	$\mathcal{O}(n + 1/\epsilon^2)$	[7]

## 2 Hardness Analysis

The ISSP is generally NP-hard, since it contains the SSP as a special case which is NP-hard. In this section, we first show that the ISSP is easier to solve than the 0-1 KP by proving that the ISSP can be equivalently reformulated as a 0-1 KP. Then, we identify some easy subclasses of the ISSP which can be solved in polynomial time (to global optimality) and therefore clearly delineate the set of computationally tractable problems within the general class of NP-hard ISSPs.

Without loss of generality, we make the following assumption on the inputs of the ISSP throughout this paper.

**Assumption 1** *The inputs of ISSP (1.3) satisfies  $T > \max_{1 \leq i \leq n} a_{i,2}$ .*

If Assumption 1 is not satisfied, there must exist an index  $i$  such that  $T \leq a_{i,2}$ . In this case, we can either find the solution of the ISSP which is  $x_i = T$  and  $x_j = 0$  for all  $j \neq i$  (if  $T \geq a_{i,1}$ ), or remove the corresponding interval  $[a_{i,1}, a_{i,2}]$  without losing any optimality (if  $T < a_{i,1}$ ).

Next we give an equivalent reformulation of the ISSP. The variables  $\{x_i\}_{i=1}^n$  in the ISSP are often called semi-continuous<sup>3</sup> [14]. Problems with semi-continuous variables can be equivalently transformed into a mixed integer program by introducing some auxiliary variables. Specifically, we can introduce binary variables  $\{y_i\}_{i=1}^n$  and real variables  $\{z_i\}_{i=1}^n$  satisfying  $0 \leq z_i \leq (a_{i,2} - a_{i,1})y_i$ ,  $i = 1, 2, \dots, n$ . Then, for any  $i = 1, 2, \dots, n$ , it is simple to verify

$$x_i \in \{0\} \cup [a_{i,1}, a_{i,2}] \iff x_i = a_{i,1}y_i + z_i, \quad y_i \in \{0, 1\}, \quad 0 \leq z_i \leq (a_{i,2} - a_{i,1})y_i.$$

Therefore, ISSP (1.3) can be equivalently reformulated as

$$\begin{aligned} & \max_{y, z} \sum_{i=1}^n (a_{i,1}y_i + z_i) \\ & \text{s.t.} \quad \sum_{i=1}^n (a_{i,1}y_i + z_i) \leq T, \\ & \quad y_i \in \{0, 1\}, \quad i = 1, 2, \dots, n, \\ & \quad 0 \leq z_i \leq (a_{i,2} - a_{i,1})y_i, \quad i = 1, 2, \dots, n. \end{aligned} \tag{2.1}$$

We can further eliminate the variables  $\{z_i\}_{i=1}^n$  in the above problem (2.1) and transform it into an equivalent problem with only binary variables.

**Theorem 2.1** *ISSP (2.1) is equivalent to*

$$\begin{aligned} & \max_y \min \left\{ \sum_{i=1}^n a_{i,2}y_i, T \right\} \\ & \text{s.t.} \quad \sum_{i=1}^n a_{i,1}y_i \leq T, \\ & \quad y_i \in \{0, 1\}, \quad i = 1, 2, \dots, n. \end{aligned} \tag{2.2}$$

Theorem 2.1 builds a bridge between the ISSP and the 0-1 KP. It shows that any algorithms designed for the 0-1 KP

$$\begin{aligned} & \max_y \sum_{i=1}^n a_{i,2}y_i \\ & \text{s.t.} \quad \sum_{i=1}^n a_{i,1}y_i \leq T, \\ & \quad y_i \in \{0, 1\}, \quad i = 1, 2, \dots, n \end{aligned} \tag{2.3}$$

can be used to solve the corresponding ISSP and thus the ISSP is easier than the 0-1 KP. In particular, if the optimal value of problem (2.3) is greater than

---

<sup>3</sup> Strictly speaking, the variables  $\{x_i\}_{i=1}^n$  in the ISSP are not semi-continuous, since  $x_i$  in the ISSP can be either zero or integers in the interval  $[a_{i,1}, a_{i,2}]$  while the semi-continuous variable  $x_i$  can be zero or any continuous value in the corresponding interval. However, as will become clear soon, the intrinsic difficulty of solving the ISSP lies in determining whether  $x_i$  should be zero or belong to  $[a_{i,1}, a_{i,2}]$ . Once this is done, it is simple to obtain an optimal solution of the ISSP. Therefore, we actually can drop the constraint  $x_i \in \mathbb{Z}$  in the ISSP.

or equal to  $T$ , we can obtain a solution of the ISSP from the solution of problem (2.3) (see Case A of the proof of Theorem 2.1 in Appendix A); otherwise, the two problems share the same solution. Table 1.1 summarizes some known FPTASs for the SSP, the ISSP, and the 0-1 KP, which are consistent with the above analysis, i.e., the difficulty of the ISSP lies between the SSP and the 0-1 KP. Theorem 2.1 also implies that any subclass of the ISSP is polynomial time solvable if the corresponding 0-1 KP problem (2.3) is polynomial time solvable.

Now, we present some polynomial time solvable subclasses of the ISSP.

**Theorem 2.2** *Suppose the inputs of the ISSP satisfy*

$$T \geq \left\lceil \frac{\max_{1 \leq i \leq n} \{a_{i,1}\}}{\min_{1 \leq i \leq n} \{a_{i,2} - a_{i,1}\}} \right\rceil \max_{1 \leq i \leq n} \{a_{i,1}\}. \quad (2.4)$$

*Then the ISSP is polynomial time solvable.*

**Theorem 2.3** *Suppose*

$$a_{i,2} \geq c a_{i,1}, \quad i = 1, 2, \dots, n \quad (2.5)$$

*holds true for some  $c > 1$  and  $T$  obeys the uniform distribution over the interval  $\left( \max_{1 \leq i \leq n} a_{i,2}, \sum_{i=1}^n a_{i,2} \right]$ . Then, the probability that the ISSP is polynomial time solvable is at least  $\min \left\{ 1, 2 \left( 1 - \frac{1}{c} \right) \right\}$ . Moreover, if (2.5) holds true with  $c \geq 2$ , then ISSP (1.3) is polynomial time solvable.*

Theorems 2.2 and 2.3 identify two subclasses of the ISSP which are polynomial time solvable. They essentially indicate that the ISSP is polynomial time solvable if the length of the intervals are sufficiently large. In particular, Theorem 2.2 shows that the ISSP is polynomial time solvable if  $a_{i,2} - a_{i,1}$  is sufficiently large for all  $i$  and Theorem 2.3 shows that the ISSP is polynomial time solvable if  $a_{i,2}/a_{i,1}$  is sufficiently large for all  $i$ . These results are consistent with our intuition, since there is more freedom to pick an element in a large interval.

### 3 A New FPTAS for the ISSP

In this section, we propose a new FPTAS for the ISSP and analyze its time and space complexities. More specifically, we first give a new solution structure of the ISSP in Section 3.1. By the use of this new solution property, we propose a new FPTAS for the ISSP. Since the proposed FPTAS is rather lengthy and complicated, we first give a high level preview of it in Section 3.2 and then provide a technical description of it in Section 3.3. Finally, we analyze the time and space complexities of the proposed FPTAS in Section 3.4.

### 3.1 A New Solution Structure

To present the new structure, we first review some existing results of the solution of the ISSP in [8].

**Definition 3.1** ([8]) For any feasible solution  $\{x_i\}_{i=1}^n$  of ISSP (1.3), if  $x_i = a_{i,1}$ , then  $[a_{i,1}, a_{i,2}]$  is called the left anchored interval; if  $x_i = a_{i,2}$ , then  $[a_{i,1}, a_{i,2}]$  is called the right anchored interval; if  $x_i \in (a_{i,1}, a_{i,2})$ , then  $[a_{i,1}, a_{i,2}]$  is called the midrange interval.

**Lemma 3.2** ([8]) *The ISSP has an optimal solution with at most one midrange interval.*

**Lemma 3.3** ([8]) *The ISSP has an optimal solution with the following property: if the optimal solution has one midrange interval, then all left anchored intervals precede the midrange interval and all right anchored intervals follow the midrange interval.*

Now, we present the new solution structure of the ISSP.

**Lemma 3.4** *Suppose the inputs of the ISSP satisfy*

$$a_{1,2} - a_{1,1} \leq a_{2,2} - a_{2,1} \leq \dots \leq a_{n,2} - a_{n,1}.$$

*Then the ISSP has an optimal solution with the following property: if  $x_j$  is the midrange element in the solution, then neither the left anchored intervals nor the right anchored intervals follow  $[a_{j,1}, a_{j,2}]$ .*

By exploiting the special structure of the solution of the ISSP in Lemma 3.3, an FPTAS was proposed by Kothari *et al.* in [8]. The proposed FPTAS in this paper is based on the new solution structure of the ISSP in Lemma 3.4. The time and space complexities of the two FPTASs can be found in Table 1.1.

### 3.2 High Level Preview of the Proposed FPTAS

In this subsection, we give a high level preview of the proposed FPTAS. Before doing that, we first present a pseudo-polynomial time dynamic programming algorithm (Algorithm 3.2) for solving the ISSP. Algorithm 3.2 exploits the special solution structure of the ISSP in Lemma 3.4 and its basic idea is to enumerate the possible midrange interval. More specifically, in Algorithm 3.2, the set  $\Delta_i^*$  in line 12 contains all values

$$\left\{ \sum_{j \in \mathcal{J}_i} x_j \mid \sum_{j \in \mathcal{J}_i} x_j \leq T, x_j \in \{a_{j,1}, a_{j,2}\}, \mathcal{J}_i \subset \{1, 2, \dots, i\} \right\}$$

and the only possible midrange interval is  $[a_{m,1}, a_{m,2}]$  (see line 7), where  $m$  is the smallest one achieving the maximum value among

$$\left\{ \max\{\delta \in \Delta_{i-1}^* \mid \delta \leq T - a_{i,1}\} + a_{i,2} \right\}_{i=1}^n.$$

In particular, if

$$\max\{\delta \in \Delta_{m-1}^* \mid \delta \leq T - a_{m,1}\} + a_{m,2} \geq T,$$

then the target  $T$  is achievable. In this case, there is no need to compute  $\{\Delta_i^*\}_{i=m}^n$  (see line 10). Once the only possible midrange interval  $[a_{m,1}, a_{m,2}]$  is found, the optimal solution  $\{x_i^*\}_{i=1}^n$  of the ISSP can be obtained as follows: the optimal solution  $\{x_i^*\}_{i=1}^{m-1}$ , which contributes to generate  $\delta^*$  in line 14, can be found by a simple backtracking procedure (see line 15); the optimal solution  $x_m^*$  is set to be  $\min\{a_{m,2}, T - \delta^*\}$  (see line 16); and the optimal solution  $\{x_i^*\}_{i=m+1}^n$  are set to be zero (see line 17). Therefore, Algorithm 3.2 returns an optimal solution satisfying the property in Lemma 3.4. An illustration how Algorithm 3.2 works is given in Appendix B. Although Algorithm 3.2 can solve the ISSP to global optimality, both of its time complexity (of computing all values in  $\{\Delta_i^*\}_{i=1}^n$ ) and space complexity (of storing them) are  $\mathcal{O}(nT)$ .

---

**Algorithm 3.2: A Pseudo-Polynomial Time Algorithm for the ISSP**

---

**Input:** a set of intervals  $\Lambda = \{[a_{i,1}, a_{i,2}]\}_{i=1}^n$  and a target value  $T$   
**Output:** the optimal solution  $\{x_i^*\}_{i=1}^n$  and the optimal value  $T^*$   
1: sort the intervals such that  $a_{i,2} - a_{i,1} \leq a_{i+1,2} - a_{i+1,1}$ ,  $i = 1, 2, \dots, n-1$   
2:  $T^* \leftarrow 0$ ,  $\Delta_0^* \leftarrow \emptyset$  (here  $\emptyset$  is the empty set)  
3: **for**  $i = 1, 2, \dots, n$  **do**  
4:    $\delta^* \leftarrow \max\{\delta \in \Delta_{i-1}^* \mid \delta \leq T - a_{i,1}\}$  (here we define  $\max \emptyset = 0$ )  
5:   **if**  $\min\{\delta^* + a_{i,2}, T\} > T^*$  **then**  
6:      $T^* \leftarrow \min\{\delta^* + a_{i,2}, T\}$   
7:      $m \leftarrow i$   
8:   **end if**  
9:   **if**  $T^* = T$  **then**  
10:     go to line 14  
11:   **end if**  
12:    $\Delta_i^* \leftarrow \left( \{\delta + a_{i,1}, \delta + a_{i,2} \mid \delta \in \Delta_{i-1}^*\} \cup \{a_{i,1}, a_{i,2}\} \cup \Delta_{i-1}^* \right) \cap (0, T]$   
13: **end for**  
14:  $\delta^* \leftarrow \max\{\delta \in \Delta_{m-1}^* \mid \delta \leq T - a_{m,1}\}$   
15: backtrack to find a subset of  $\{a_{i,1}, a_{i,2}\}_{i=1}^{m-1}$  which contributes to generate  $\delta^*$ , and then construct the solution  $\{x_i^*\}_{i=1}^{m-1}$   
16:  $x_m^* \leftarrow \min\{a_{m,2}, T - \delta^*\}$   
17:  $x_i^* \leftarrow 0$ ,  $i = m+1, \dots, n$   
18: **return**  $T^*$  and  $\{x_i^*\}_{i=1}^n$

---

Next, we give a high level preview of the proposed FPTAS (Algorithm 3.3), which can be obtained by doing some nontrivial modifications to the above Algorithm 3.2. We remark that the proposed FPTAS for the ISSP can be used to solve the SSP.

One main modification made to Algorithm 3.2 is to partition the interval  $(0, T]$  into finitely many subintervals (depending on the given relative error  $\epsilon$ ) and to store only the smallest and largest values lying in the subintervals at each iteration. More specifically, for any given relative error  $\epsilon > 0$ , Algorithm 3.3 partitions the interval  $(0, T]$  into  $l = \lceil 1/\epsilon \rceil$  subintervals

$$I_1 = (0, \epsilon T], I_2 = (\epsilon T, 2\epsilon T], \dots, I_l = ((l-1)\epsilon T, T]$$



and only stores the smallest value  $\delta^-(k)$  and the largest value  $\delta^+(k)$  in each subinterval  $I_k$ ,  $k = 1, 2, \dots, l$ ; see lines 17 – 22. This is in sharp contrast to Algorithm 3.2, where all values in  $\{\Delta_i^*\}_{i=1}^n$  (if the target  $T$  is not achievable) or all values in  $\{\Delta_i^*\}_{i=1}^{m-1}$  (if the target  $T$  is achievable and  $[a_{m,1}, a_{m,2}]$  is the only possible midrange interval) are stored. Lemma 3.5 and Corollary 3.6 show that doing so will not lose much optimality. If we are only interested in obtaining an approximate objective value but not in getting the corresponding solution set, we can run Algorithm 3.3 without line 27. Line 27 of Algorithm 3.3 aims at recovering a corresponding solution set.

According to lines 14–23 of Algorithm 3.3, for each  $\delta \in \{\delta^-(k), \delta^+(k)\}_{k=1}^l$ , there exist  $\delta'$  and  $a_{i,j}$  such that  $\delta = \delta' + a_{i,j}$ , where  $\delta'$  is either zero or a summation of end points of some subset of  $\{[a_{k,1}, a_{k,2}]\}_{k=1}^{i-1}$  and  $a_{i,j}$  is the last item contributed to generate  $\delta$ . Obviously, to recover the approximate solution, it is useful to store such  $i$  and  $j$ . These indices are stored in  $d_1(\cdot) \in \{1, 2, \dots, n\}$  and  $d_2(\cdot) \in \{1, 2\}$  in the procedure **relaxed dynamic programming**. Therefore, one natural way of recovering the approximate solution is to simply backtrack the elements in  $\{\delta^-(k), \delta^+(k)\}_{k=1}^l$  with the help of  $d_1(\cdot)$  and  $d_2(\cdot)$ . However, there is a potential problem with such a simple backtracking procedure, since an element in the subinterval  $I_k$  which generates an element in the interval  $I_j$  with  $j > k$  might be updated after considering the item with the index  $d_1(j)$ . For instance, the above  $\delta'$  which contributes to generate  $\delta$  might be updated, and thus  $\delta'$  does not belong to  $\{\delta^-(k), \delta^+(k)\}_{k=1}^l$  any more. Therefore, the simple backtracking procedure does not work and it is necessary to make a modification to it in order to reconstruct a correct approximate solution.

Algorithm 3.3 overcomes the previously mentioned backtracking problem by performing the procedure **divide and conquer**. Once the procedure **backtracking** can not continue, the procedure **divide and conquer** splits the task of constructing an approximate solution with inputs  $\tilde{A}$  and  $\tilde{T}$  into two subsets  $\tilde{A}_1$  and  $\tilde{A}_2$  of (almost) the same cardinality. The procedure **relaxed dynamic programming** is then performed for both item sets independently with the target value  $\tilde{T}$ , which returns four reduced achievable arrays  $\delta_1^-(\cdot)$ ,  $\delta_1^+(\cdot)$ ,  $\delta_2^-(\cdot)$ ,  $\delta_2^+(\cdot)$  and the associated interval and end point sets  $d_{1,1}(\cdot)$ ,  $d_{1,2}(\cdot)$ ,  $d_{2,1}(\cdot)$ ,  $d_{2,2}(\cdot)$ . Lemma 3.7 guarantees that there exist  $u_1 \in \{0, \delta_1^-(\cdot), \delta_1^+(\cdot)\}$  and  $u_2 \in \{0, \delta_2^-(\cdot), \delta_2^+(\cdot)\}$  such that  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ .

To find the approximate solution corresponding to values  $u_1$  and  $u_2$  in the above, the procedure **backtracking** is first performed for the item set  $\tilde{A}_1$  with the target value  $\tilde{T} - u_2$ , which reconstructs a part of the solution contributed by  $\tilde{A}_1$  with value  $y_1^B$ . In addition, the procedure **backtracking** also records the items that have been considered to reconstruct the approximate solution. These items are collected in the set  $\Lambda^E$  and will not be considered any more. Obviously, by doing so, no item will appear twice in the approximate solution. Lemma 3.8 states that this will not lose any optimality in terms of reconstructing an approximate solution. If  $y_1^B$  is not sufficiently close to  $\tilde{T} - u_2$ , a recursive execution of the procedure **divide and conquer** for the item set  $\tilde{A}_1 \setminus \Lambda^E$  with the target value  $\tilde{T} - u_2 - y_1^B$  finally reconstructs a solution with the value  $y_1^B + y_1^{DC}$ , which is close enough to  $u_1$ . The same can be applied to the item set  $\tilde{A}_2$  with the target value  $\tilde{T} - y_1^B - y_1^{DC}$ , which reconstructs a solution with the value  $y_2^B + y_2^{DC}$ . Lemma 3.9

shows that the returned approximate value  $y^{DC} = y_1^B + y_1^{DC} + y_2^B + y_2^{DC}$  satisfies  $\tilde{T} - \epsilon T \leq y^{DC} \leq \tilde{T}$ .

Since each execution of the procedure **divide and conquer** returns at least one item for the solution by backtracking, the depth of its recursion is bounded by  $\mathcal{O}(\log n)$ . Therefore, Algorithm 3.3 will terminate after (totally) recursively calling the procedure **divide and conquer**  $n$  times. Theorems 3.10 and 3.11 show that Algorithm 3.3 is indeed an FPTAS for the ISSP with time and space complexities being  $\mathcal{O}(n \max\{1/\epsilon, \log n\})$  and  $\mathcal{O}(n + 1/\epsilon)$ , respectively.

### 3.3 Technical Description of the Proposed FPTAS

In this subsection, we describe the proposed FPTAS for the ISSP in a technical fashion, where we use  $I_k$  to denote the  $k$ -th subinterval, use  $\delta^-(k)$  and  $\delta^+(k)$  to denote the smallest and largest values in  $I_k$ , and use  $\Lambda^E$  to denote the set of intervals to be removed during the execution of the algorithm. Algorithm 3.3 below is the proposed FPTAS for the ISSP, which calls the procedure **divide and conquer**. The procedure **divide and conquer** further calls the procedures **relaxed dynamic programming** and **backtracking** to construct the approximate solution.

In Algorithm 3.3, only the largest value  $\delta^+(k)$  and the smallest value  $\delta^-(k)$  in each subinterval  $k = 1, 2, \dots, l := \lceil 1/\epsilon \rceil$  are stored; see lines 17 – 22. Lemma 3.5 and Corollary 3.6 show that this will not lose much optimality. It will become clear from Theorem 3.10 and the discussions below it why the the last input of the procedure **divide and conquer** is set to be  $\min\{\hat{\delta} + \epsilon T, T - a_{m,1}\}$  in line 27. By doing so, Algorithm 3.3 is able to return an (exactly) optimal solution of the ISSP when  $\hat{\delta} + \epsilon T \leq T - a_{m,1}$ .

The procedure **divide and conquer**, with inputs  $(\tilde{\Lambda}, \tilde{T})$ , aims at finding an approximate solution  $\{\hat{x}_i\}_{i=1}^n$  and the approximate value  $y^{DC}$  from the given subset of intervals  $\tilde{\Lambda} \subseteq \{[a_{i,1}, a_{i,2}]\}_{i=1}^n$  such that  $\tilde{T} - \epsilon T \leq y^{DC} \leq \tilde{T}$ . Lemma 3.9 establishes that the returned approximate value  $y^{DC} = y_1^B + y_1^{DC} + y_2^B + y_2^{DC}$  indeed satisfies  $\tilde{T} - \epsilon T \leq y^{DC} \leq \tilde{T}$ .

The procedure **relaxed dynamic programming**, with inputs  $(\tilde{\Lambda}, \tilde{T})$ , is a subroutine used in the procedure **divide and conquer** to recover the approximate solution. In the procedure **relaxed dynamic programming**, besides the largest value  $\delta^+(k)$  and the smallest value  $\delta^-(k)$  in each subinterval  $k = 1, 2, \dots, l := \lceil \tilde{T}/(\epsilon T) \rceil$ , the index and the end point of the last interval which generates  $\delta \in \{\delta^+(k), \delta^-(k)\}_{k=1}^{\tilde{l}}$  are also stored in  $d_1(\cdot) \in \{1, 2, \dots, \tilde{l}\}$  and  $d_2(\cdot) \in \{1, 2\}$ . These information will be used in the procedure **backtracking** to recover the approximate solution.

Taking the outputs of the procedure **relaxed dynamic programming**  $\delta^-(\cdot)$ ,  $\delta^+(\cdot)$ ,  $d_1(\cdot)$ , and  $d_2(\cdot)$  as inputs, the procedure **backtracking** backtracks the intervals which contribute to generate the largest value in  $\{\delta^+(j), \delta^-(j) \mid \delta^+(j) \leq \tilde{T}, \delta^-(j) \leq \tilde{T}\}$  (see line 1) with the help of  $d_1(\cdot)$  and  $d_2(\cdot)$  and reconstructs a partial approximate solution. The procedure **backtracking** stops if for some  $u \in I_k$ , the solution value conditions  $\delta^+(k) + y \leq \tilde{T}$  and  $\delta^-(k) + y \geq \tilde{T} - \epsilon T$  do not hold true, or  $\delta^+(k) + y \leq \tilde{T}$  holds true but  $u$  is updated by some later interval

**Algorithm 3.3: A New FPTAS for the ISSP**


---

**Input:** a set of intervals  $\Lambda = \{[a_{i,1}, a_{i,2}]\}_{i=1}^n$ , the target value  $T$ , and the relative approximation error  $\epsilon$

**Output:** the approximate solution  $\{x_i^A\}_{i=1}^n$  and the approximate value  $T^A$

- 1: sort the intervals such that  $a_{i,2} - a_{i,1} \leq a_{i+1,2} - a_{i+1,1}$ ,  $i = 1, 2, \dots, n-1$
- 2:  $\hat{T} \leftarrow 0$ ,  $\hat{\delta} \leftarrow 0$
- 3:  $\delta^+(i) \leftarrow 0$ ,  $\delta^-(i) \leftarrow 0$ ,  $i = 1, 2, \dots, l := \lceil \frac{1}{\epsilon} \rceil$
- 4: **for**  $i = 1, 2, \dots, n$  **do**
- 5:    $\bar{\delta} \leftarrow \max_{1 \leq k \leq l} \{\delta^-(k), \delta^+(k) \mid \delta^-(k), \delta^+(k) \leq T - a_{i,1}\}$
- 6:   **if**  $\min\{\bar{\delta} + a_{i,2}, T\} > \hat{T}$  **then**
- 7:      $\hat{T} \leftarrow \min\{\bar{\delta} + a_{i,2}, T\}$
- 8:      $\hat{\delta} \leftarrow \bar{\delta}$
- 9:      $m \leftarrow i$
- 10:   **end if**
- 11:   **if**  $T^* = \hat{T}$  **then**
- 12:     go to line 25
- 13:   **end if**
- 14:    $\tilde{\Delta} \leftarrow (\{\delta^-(k) + a_{i,1}, \delta^+(k) + a_{i,1}, \delta^-(k) + a_{i,2}, \delta^+(k) + a_{i,2}\}_{k=1}^l \cup \{a_{i,1}, a_{i,2}\}) \cap (0, T]$
- 15:   **for** each  $\tilde{\delta} \in \tilde{\Delta}$  **do**
- 16:     find  $I_k$  that contains  $\tilde{\delta}$
- 17:     **if**  $\tilde{\delta} < \delta^-(k)$  or  $\delta^-(k) = 0$  **then**
- 18:        $\delta^-(k) \leftarrow \tilde{\delta}$
- 19:     **end if**
- 20:     **if**  $\tilde{\delta} > \delta^+(k)$  or  $\delta^+(k) = 0$  **then**
- 21:        $\delta^+(k) \leftarrow \tilde{\delta}$
- 22:     **end if**
- 23:   **end for**
- 24: **end for**
- 25:  $\Lambda^E \leftarrow \{[a_{i,1}, a_{i,2}] \mid m \leq i \leq n\}$
- 26:  $x_i^A \leftarrow 0$ ,  $i = 1, \dots, m-1$
- 27: call the procedure **divide and conquer**  $(\Lambda \setminus \Lambda^E, \min\{\hat{\delta} + \epsilon T, T - a_{m,1}\})$  for the approximate solution  $\{x_i^A\}_{i=1}^{m-1}$  and the approximate value  $\hat{T}^A$
- 28:  $x_m^A \leftarrow \min\{a_{m,2}, T - \hat{T}^A\}$
- 29:  $x_i^A \leftarrow 0$ ,  $i = m+1, \dots, n$
- 30:  $T^A \leftarrow \hat{T}^A + x_m^A$
- 31: **return**  $T^A$  and  $\{x_i^A\}_{i=1}^n$

---

with index  $d_1(\delta^+(k)) \geq i$ ; or  $\delta^-(k) + y \geq \tilde{T} - \epsilon T$  holds true but  $u$  is updated by some later interval with index  $d_1(\delta^-(k)) \geq i$ . Actually, lines 9 – 18 are to speed up the procedure **backtracking**. It will become clear from Lemma 3.8 and the discussions below it that Algorithm 3.3 can still recover an approximate solution of the ISSP if there are no lines 9 – 18 in the procedure **backtracking**.

An illustration how Algorithm 3.3 works is given in Appendix B.

### 3.4 Worst-Case Time and Space Complexity Analysis

For simplicity of our analysis, we introduce the following notation. Let  $\Delta_{\tilde{\Lambda}}^*$  denote the full dynamic programming arrays computed from the intervals in  $\tilde{\Lambda}$ , and  $\Delta_{\tilde{\Lambda}}$  the relaxed dynamic programming arrays computed from the intervals in  $\tilde{\Lambda}$ . In particular, when  $\tilde{\Lambda} = \{1, 2, \dots, i\}$ , we denote them by  $\Delta_i^*$  and  $\Delta_i$ , respectively.

---

**Procedure divide and conquer**  $(\tilde{A}, \tilde{T})$ **Input:** a subset of intervals  $\tilde{A} \subseteq \{[a_{i,1}, a_{i,2}]\}_{i=1}^n$  and the target value  $\tilde{T}$ **Output:** the updated approximate solution  $\{\hat{x}_i\}_{i=1}^n$  and the approximate value  $y^{DC}$ 

- 1: split  $\tilde{A}$  into  $\tilde{A}_1$  and  $\tilde{A}_2$ , which contain  $\left\lceil \frac{|\tilde{A}|}{2} \right\rceil$  and  $\left\lfloor \frac{|\tilde{A}|}{2} \right\rfloor$  elements respectively
  - 2: call **relaxed dynamic programming**  $(\tilde{A}_1, \tilde{T})$  to obtain  $\delta_1^-(\cdot), \delta_1^+(\cdot), d_{1,1}(\cdot), d_{1,2}(\cdot)$
  - 3: call **relaxed dynamic programming**  $(\tilde{A}_2, \tilde{T})$  to obtain  $\delta_2^-(\cdot), \delta_2^+(\cdot), d_{2,1}(\cdot), d_{2,2}(\cdot)$
  - 4: find  $u_1 \in \{0, \delta_1^-(\cdot), \delta_1^+(\cdot)\}$  and  $u_2 \in \{0, \delta_2^-(\cdot), \delta_2^+(\cdot)\}$  such that  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$  (Lemma 3.7 guarantees the existence of such  $u_1$  and  $u_2$ )
  - 5:  $y_1^B \leftarrow 0, y_1^{DC} \leftarrow 0, y_2^B \leftarrow 0, y_2^{DC} \leftarrow 0$
  - 6: **if**  $\tilde{T} - u_2 > \epsilon T$  **then**
  - 7:   call **backtracking**  $(\delta_1^-(\cdot), \delta_1^+(\cdot), d_{1,1}(\cdot), d_{1,2}(\cdot), \tilde{A}_1, \tilde{T} - u_2)$  to obtain  $y_1^B$  and  $\Lambda^E$
  - 8: **end if**
  - 9: **if**  $\tilde{T} - u_2 - y_1^B > \epsilon T$  **then**
  - 10:   call **divide and conquer**  $(\tilde{A}_1 \setminus \Lambda^E, \tilde{T} - u_2 - y_1^B)$  to obtain  $y_1^{DC}$
  - 11: **end if**
  - 12: **if**  $\tilde{T} - y_1^B - y_1^{DC} > \epsilon T$  **then**
  - 13:   call **relaxed dynamic programming**  $(\tilde{A}_2, \tilde{T} - y_1^B - y_1^{DC})$  to obtain  $\delta_2^-(\cdot), \delta_2^+(\cdot), d_{2,1}(\cdot), d_{2,2}(\cdot)$
  - 14:   call **backtracking**  $(\delta_2^-(\cdot), \delta_2^+(\cdot), d_{2,1}(\cdot), d_{2,2}(\cdot), \tilde{A}_2, \tilde{T} - y_1^B - y_1^{DC})$  to obtain  $y_2^B$  and  $\Lambda^E$
  - 15: **end if**
  - 16: **if**  $\tilde{T} - y_1^B - y_1^{DC} - y_2^B > \epsilon T$  **then**
  - 17:   call **divide and conquer**  $(\tilde{A}_2 \setminus \Lambda^E, \tilde{T} - y_1^B - y_1^{DC} - y_2^B)$  to obtain  $y_2^{DC}$
  - 18: **end if**
  - 19:  $y^{DC} \leftarrow y_1^B + y_1^{DC} + y_2^B + y_2^{DC}$
  - 20: **return**  $y^{DC}$
- 

---

**Procedure relaxed dynamic programming**  $(\tilde{A}, \tilde{T})$ **Input:** a subset of intervals  $\tilde{A} \subseteq \{[a_{i,1}, a_{i,2}]\}_{i=1}^n$  and the target value  $\tilde{T}$ **Output:** dynamic programming arrays  $\delta^-(\cdot), \delta^+(\cdot), d_1(\cdot), d_2(\cdot)$ 

- 1:  $\delta^+(i) \leftarrow 0, \delta^-(i) \leftarrow 0, i = 1, 2, \dots, \tilde{l} := \left\lceil \tilde{T}/(\epsilon T) \right\rceil$
  - 2: **for** each  $i \in \{i \mid [a_{i,1}, a_{i,2}] \in \tilde{A}\}$  **do**
  - 3:   **for**  $j = 1, 2$  **do**
  - 4:      $\tilde{\Delta} \leftarrow \left( \{\delta^-(k) + a_{i,j}, \delta^+(k) + a_{i,j}\}_{k=1}^{\tilde{l}} \cup \{a_{i,j}\} \right) \cap (0, \tilde{T}]$
  - 5:     **for** each  $\tilde{\delta} \in \tilde{\Delta}$  **do**
  - 6:       find  $I_k$  that contains  $\tilde{\delta}$
  - 7:       **if**  $\tilde{\delta} < \delta^-(k)$  or  $\delta^-(k) = 0$  **then**
  - 8:          $\delta^-(k) \leftarrow \tilde{\delta}, d_1(\delta^-(k)) \leftarrow i, d_2(\delta^-(k)) \leftarrow j$
  - 9:       **end if**
  - 10:       **if**  $\tilde{\delta} > \delta^+(k)$  or  $\delta^+(k) = 0$  **then**
  - 11:          $\delta^+(k) \leftarrow \tilde{\delta}, d_1(\delta^+(k)) \leftarrow i, d_2(\delta^+(k)) \leftarrow j$
  - 12:       **end if**
  - 13:     **end for**
  - 14:   **end for**
  - 15: **end for**
  - 16: **return**  $\delta^-(\cdot), \delta^+(\cdot), d_1(\cdot), d_2(\cdot)$
-

---

**Procedure backtracking** ( $\delta^-(\cdot), \delta^+(\cdot), d_1(\cdot), d_2(\cdot), \tilde{\Lambda}, \tilde{T}$ )

**Input:** dynamic programming arrays  $\delta^-(\cdot), \delta^+(\cdot), d_1(\cdot), d_2(\cdot)$ , a subset of intervals  $\tilde{\Lambda} \subseteq \{[a_{i,1}, a_{i,2}]\}_{i=1}^n$ , and the target value  $\tilde{T}$ 
**Output:** the updated approximate solution  $\{\hat{x}_i\}_{i=1}^n$ , the set of intervals  $\Lambda^E$ , and the partial approximate value  $y$ 

```

1:  $u \leftarrow \max\{\delta^+(j), \delta^-(j) \mid \delta^+(j) \leq \tilde{T}, \delta^-(j) \leq \tilde{T}\}$ 
2:  $y \leftarrow 0$ 
3: repeat
4:    $i \leftarrow d_1(u), j \leftarrow d_2(u)$ 
5:    $\hat{x}_i \leftarrow a_{i,j}$ 
6:    $\Lambda^E \leftarrow \Lambda^E \cup \{[a_{k,1}, a_{k,2}] \in \tilde{\Lambda} \mid k \geq i\}$ 
7:    $y \leftarrow y + a_{i,j}$ 
8:    $u \leftarrow u - a_{i,j}$ 
9:   if  $u > 0$  then
10:    find  $I_k$  that contains  $u$ 
11:    if  $\delta^+(k) + y \leq \tilde{T}$  and  $d_1(\delta^+(k)) < i$  then
12:       $u \leftarrow \delta^+(k)$ 
13:    else if  $\delta^-(k) + y \geq \tilde{T} - \epsilon T$  and  $d_1(\delta^-(k)) < i$  then
14:       $u \leftarrow \delta^-(k)$ 
15:    else
16:       $u \leftarrow 0$ 
17:    end if
18:  end if
19: until  $u = 0$ 
20: return  $y$  and  $\Lambda^E$ 

```

---

Throughout this section, we shall also use the term “ $\Delta_{\tilde{\Lambda}}^*$  associated with a target  $\tilde{T}$ ” to denote the set  $\{\delta \mid \delta \in \Delta_{\tilde{\Lambda}}^*, \delta \leq \tilde{T}\}$ . The same applies to  $\Delta_{\tilde{\Lambda}}$ ,  $\Delta_i^*$ , and  $\Delta_i$ .

With the above notation, it is obvious to see that  $\{\delta \in \Delta_n^* \mid \delta \leq T\}$  is the optimal value of the ISSP. According to lines 14 – 22 of Algorithm 3.3, we know  $\Delta_i \subseteq \Delta_i^*$  for all  $i = 1, 2, \dots, n$  and  $\Delta_{\tilde{\Lambda}} \subseteq \Delta_{\tilde{\Lambda}}^*$  for all  $\tilde{\Lambda} \subseteq \{1, 2, \dots, n\}$ .

The next lemma (Lemma 3.5) plays a fundamental role in analyzing the proposed FPTAS for the ISSP. It says that each element computed by the full dynamic programming procedure can be well approximated (with a bounded error) by some element computed by the relaxed dynamic programming procedure. More specifically, Lemma 3.5 shows that, for any  $\delta \in \Delta_{\tilde{\Lambda}}^*$ , there exists an  $\underline{\delta} \in \Delta_{\tilde{\Lambda}}$  such that  $0 \leq \delta - \underline{\delta} \leq \epsilon T$ .

**Lemma 3.5** *For each  $\delta \in \Delta_i^*$  associated with the target  $\tilde{T}$ , one of the following two statements is true:*

1. *There exist  $\underline{\delta}, \bar{\delta} \in \Delta_i$  such that  $\underline{\delta} \leq \delta \leq \bar{\delta}$  and  $\bar{\delta} - \underline{\delta} \leq \epsilon T$ .*
2. *There exists  $\underline{\delta} \in \Delta_i$  such that  $\tilde{T} - \epsilon T \leq \underline{\delta} \leq \delta \leq \tilde{T}$ .*

By setting  $\delta$  to be the optimal value of the ISSP in Lemma 3.5, we have the following corollary.

**Corollary 3.6** *Suppose  $\delta^*$  is the optimal value of the ISSP with inputs  $\tilde{\Lambda}$  and  $\tilde{T}$ . Then, there exists  $\delta \in \Delta_{\tilde{\Lambda}}$  such that either  $\delta = \delta^*$  or  $\tilde{T} - \epsilon T \leq \delta \leq \delta^* \leq \tilde{T}$  holds true.*

The following lemma (Lemma 3.7) shows the existence of  $u_1$  and  $u_2$  in line 4 of the procedure **divide and conquer**.

**Lemma 3.7** *Suppose there exists  $\delta \in \Delta_{\tilde{A}}$  such that  $\tilde{T} - \epsilon T \leq \delta \leq \tilde{T}$  when the procedure **divide and conquer** starts. Then, after the splitting of  $\tilde{A}$  into  $\tilde{A}_1$  and  $\tilde{A}_2$ , there must exist  $u_1 \in \{0\} \cup \Delta_{\tilde{A}_1}$  and  $u_2 \in \{0\} \cup \Delta_{\tilde{A}_2}$  such that  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ .*

Lemma 3.7 establishes the existence of  $u_1 \in \{0\} \cup \Delta_{\tilde{A}_1}$  and  $u_2 \in \{0\} \cup \Delta_{\tilde{A}_2}$  such that  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ . In fact, such  $u_1$  and  $u_2$  might not be unique; see the remark in Appendix B. In our implementation of the procedure **divide and conquer**, we use the following strategy, with both time and space complexities being  $\mathcal{O}(1/\epsilon)$ , to find a pair of  $u_1$  and  $u_2$  satisfying  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ . Without loss of generality, suppose  $\Delta_{\tilde{A}_1} = \{\delta_1^1, \delta_1^2, \dots, \delta_1^{2l}\}$  and  $\Delta_{\tilde{A}_2} = \{\delta_2^1, \delta_2^2, \dots, \delta_2^{2l}\}$ , where  $\delta_1^i \leq \delta_1^{i+1}$  and  $\delta_2^i \leq \delta_2^{i+1}$  for all  $i = 1, 2, \dots, 2l - 1$  and  $l \leq \lceil 1/\epsilon \rceil$ . For convenience, we also define  $\delta_1^0 = \delta_2^0 = 0$ . Let  $i = 0$  and  $j = 2l$ ,  $u_1 = \delta_1^0$ , and  $u_2 = \delta_2^{2l}$ , then repeatedly do the following until  $u_1 + u_2 \in [\tilde{T} - \epsilon T, \tilde{T}]$ : if  $u_1 + u_2 < \tilde{T} - \epsilon T$ , let  $u_1 = \delta_1^{i+1}$  and increment  $i$  by 1; if  $u_1 + u_2 > \tilde{T}$ , let  $u_2 = \delta_2^{j-1}$  and decrement  $j$  by 1. It is simple to verify that the above strategy returns a pair of desired  $u_1$  and  $u_2$  and its time and space complexities are  $\mathcal{O}(1/\epsilon)$ .

The following lemma (Lemma 3.8) states that the procedure **backtracking** can successfully backtrack a part of the approximate solution by using the relaxed dynamic programming arrays. This is in sharp contrast to the pseudo-polynomial time algorithm (Algorithm 3.2) for the ISSP, where the solution is backtracked by using the full dynamic programming arrays. We remark that the space complexity of storing the relaxed dynamic programming arrays is  $\mathcal{O}(1/\epsilon)$  and the space complexity of storing the full dynamic programming arrays is  $\mathcal{O}(nT)$ .

**Lemma 3.8** *Suppose there exists  $\delta' \in \Delta_{\tilde{A}}^*$  such that  $\tilde{T} - \epsilon T \leq \delta' \leq \tilde{T}$  when the procedure **backtracking** starts and  $y$  is its output. Then, there exists  $\delta \in \{0\} \cup \Delta_{\tilde{A} \setminus A^E}^*$  such that  $\tilde{T} - \epsilon T \leq y + \delta \leq \tilde{T}$ .*

It is worthwhile remarking that lines 9 – 18 are for speeding up the procedure **backtracking**. More specifically, the procedure **backtracking** equipped with lines 9 – 18 could potentially backtrack more than one step. In contrast, the procedure **backtracking**, without lines 9 – 18, can only backtrack one step. Lines 9 – 18 will not affect the time and space complexities of the proposed FPTAS, which will become more clear in the following Theorem 3.11.

**Lemma 3.9** *Suppose there exists  $\delta \in \Delta_{\tilde{A}}^*$  such that  $\tilde{T} - \epsilon T \leq \delta \leq \tilde{T}$  when the procedure **divide and conquer** starts. Then, the output  $y^{DC}$  of the procedure **divide and conquer** satisfies  $\tilde{T} - \epsilon T \leq y^{DC} \leq \tilde{T}$ .*

We are now ready to present the main results of this section.

**Theorem 3.10** *Algorithm 3.3 returns an  $(1-\epsilon)$ -approximate solution of the ISSP.*

Note that in line 27 of Algorithm 3.3, the last input of the procedure **divide and conquer** is set to be  $\min\{\hat{\delta} + \epsilon T, T - a_{m,1}\}$ . This trick makes Algorithm 3.3 return an (exactly) optimal solution of the ISSP when  $\hat{\delta} + \epsilon T \leq T - a_{m,1}$ ; see Case A of the proof of Theorem 3.10 in Appendix A. Actually, the result presented in Theorem 3.10 still holds true if  $\min\{\hat{\delta} + \epsilon T, T - a_{m,1}\}$  is replaced

with  $\hat{\delta} = \min \left\{ \hat{\delta}, T - a_{m,1} \right\}$ , where the last equality is due to  $\hat{\delta} \leq T - a_{m,1}$ . However, if so, Algorithm 3.3 would not enjoy the nice property of returning the (exactly) optimal solution when  $\hat{\delta} + \epsilon T \leq T - a_{m,1}$ .

**Theorem 3.11** *The time complexity of Algorithm 3.3 is  $\mathcal{O}(n \max \{1/\epsilon, \log n\})$ , and the space complexity is  $\mathcal{O}(n + 1/\epsilon)$ .*

We compare the proposed FPTAS and the one in [8] in terms of the time and space complexities; see Table 1.1. The time complexity of the two FPTASs are comparable to each other. This is because  $\log n$  is generally smaller than  $1/\epsilon$ . However, the space complexity of the proposed FPTAS is significantly lower than the one in [8]. The significant improvement in the space complexity makes the proposed FPTAS possible to solve large scale ISSP instances on a limited memory machine, which is generally impossible for the FPTAS in [8].

#### 4 Numerical Experiments

We implemented the proposed FPTAS for solving the ISSP with C++. Our numerical experiments were done on a personal computer with Ubuntu 10.04.2 operating system, Intel Core i7 CPU, 8 GB memory, and the source code is compiled with GCC 4.4.3.

Since there is no available test set for the ISSP, we tested the proposed FPTAS on the test set for the SSP, with some modifications.

Instance A:  $a_{i,1} = a_{i,2} = 2^{k+n+1} + 2^{k+i} + 1$  and  $T = \left\lfloor \frac{1}{2} \sum_{i=1}^n a_{i,2} \right\rfloor$ , where  $k = \lfloor \log_2(n) \rfloor$ ;

Instance B:  $a_{i,1} = a_{i,2} = n(n+1) + i$  and  $T = \left\lfloor \frac{n-1}{2} \right\rfloor n(n+1) + \frac{n(n-1)}{2}$ ;

Instance C:  $a_{i,2}$  is an integer generated by uniformly randomly sampling in  $[1, 10^{14}]$ ,  $a_{i,1} = \left\lfloor \frac{a_{i,2}}{c} \right\rfloor$ , and  $T = 3 \times 10^{14}$ , where  $c \geq 1$  is a given parameter;

Instance D:  $a_{i,2}$  is an integer generated by uniformly randomly sampling in  $[1, 10^{14}]$ ,  $a_{i,1} = \left\lfloor \frac{a_{i,2}}{c_i} \right\rfloor$ , and  $T = 3 \times 10^{14}$ , where  $c_i$  is a real number generated by uniformly randomly sampling in  $[1, C]$  with a given parameter  $C \geq 1$ .

Instances A and B [3] are the SSPs. They are used to test whether the proposed FPTAS (Algorithm 3.3) for the ISSP can correctly and efficiently solve the degenerating problems. The optimal solutions of these two instances are easy to obtain but very hard to compute by the branch and bound method [3]. Instances C and D are randomly generated. They are used to test the correctness and efficiency of the proposed FPTAS for solving the ISSP.

Table 4.1 summarizes the numerical results of applying the proposed FPTAS to solve Instance A. The parameter  $n$  in Instance A can not be very large; otherwise  $a_{i,1}, a_{i,2}$ , and  $T$  will be extremely large. From Table 4.1, we can observe that the returned relative errors of all tested SSP instances are strictly less than the preselected parameter  $\epsilon$ . In particular, when  $\epsilon = 0.1\%$ , the returned relative errors of all tested SSP instances are zero, which implies that the proposed FPTAS successively solves all tested SSP instances to global optimality. These numerical results show the correctness of the proposed FPTAS for solving the SSP.

**Table 4.1** Numerical Results of Instance A

	$\epsilon = 10\%$		$\epsilon = 1\%$		$\epsilon = 0.1\%$	
$n$	relative error	time(s)	relative error	time(s)	relative error	time(s)
10	0.000%	< 0.001	0.000%	< 0.001	0.000%	< 0.001
15	1.515%	< 0.001	0.342%	< 0.001	0.000%	0.001
20	0.000%	< 0.001	0.000%	< 0.001	0.000%	0.002
25	9.616%	< 0.001	0.059%	< 0.001	0.000%	0.003
30	9.690%	< 0.001	0.000%	< 0.001	0.000%	0.004
35	5.558%	< 0.001	0.347%	< 0.001	0.000%	0.005

Table 4.2 summarizes the numerical results of applying the proposed FPTAS to solve Instance B. Again, the returned relative errors of all tested SSP instances are not greater than the given tolerance  $\epsilon$ , which means that the proposed FPTAS can solve SSP correctly as claimed in Theorem 3.10. From Table 4.2, we can observe that the computational time of the proposed FPTAS grows (roughly) linearly with  $n$  when  $\epsilon$  is fixed and also (roughly) linearly with  $1/\epsilon$  when  $n$  is fixed. This matches with the time complexity  $\mathcal{O}(n \max\{1/\epsilon, \log n\})$  of the proposed FPTAS, since  $1/\epsilon$  and  $\log n$  are comparable to each other for the tested case  $\epsilon = 10\%$  and  $1/\epsilon \gg \log n$  for the tested cases  $\epsilon = 1\%$  and  $\epsilon = 0.1\%$ . In particular, it takes the proposed FPTAS less than 0.8 seconds to solve the SSP instance with  $n = 10,000$  and  $\epsilon = 0.1\%$ . From the above numerical results, we can conclude that the proposed FPTAS can efficiently solve the SSP.

**Table 4.2** Numerical Results of Instance B

	$\epsilon = 10\%$		$\epsilon = 1\%$		$\epsilon = 0.1\%$	
$n$	relative error	time(s)	relative error	time(s)	relative error	time(s)
10	0.844%	< 0.001	0.000%	< 0.001	0.000%	< 0.001
50	8.353%	< 0.001	0.071%	0.001	0.000%	0.002
100	8.166%	< 0.001	0.019%	0.002	0.000%	0.004
500	9.637%	0.002	0.902%	0.008	0.000%	0.015
1,000	9.818%	0.001	0.851%	0.008	0.000%	0.049
5,000	9.964%	0.007	0.970%	0.042	0.080%	0.400
10,000	9.982%	0.014	0.985%	0.085	0.080%	0.795



Table 4.3 summarizes the numerical results of applying the proposed FPTAS to solve Instance C. The results in Table 4.3 are obtained by averaging over 100 randomly generated ISSP instances for each fixed  $n$  and  $c$ . The worst results among these 100 tested instances (in terms of the relative error and the computational time, respectively) are also reported in parentheses in Table 4.3. Since the optimal value of Instance C is difficult to obtain, we set it to be the target  $T$  when calculating the relative errors. The relative errors computed in this way are obviously larger than or equal to the “true” relative errors. It can be observed from Table 4.3 that the worst relative errors of all tested instances (and thus the average relative errors) are not greater than the desired relative error  $\epsilon$ , which shows the correctness of the proposed FPTAS for solving the ISSP. Table 4.3 also demonstrates the efficiency of the proposed FPTAS when applied to solve large scale instances of Instance C. The proposed FPTAS is capable of returning an approximate solution of the tested ISSP instances with  $n = 100,000$  and  $\epsilon = 0.1\%$  within 0.1 second in average.

It is worthwhile remarking that the choice of the parameter  $c$  in Instance C actually affects the efficiency of the proposed FPTAS. As we can observe from Table 4.3, as the parameter  $c$  in Instance C decreases, the average computational time of using the proposed FPTAS to solve the corresponding ISSP instances slightly increases. This becomes obvious for the ISSP instances with  $n \geq 50,000$ . The above observation is consistent with Theorem 2.3, which basically says that the ISSP becomes more difficult to solve as the parameter  $c$  there decreases.

Table 4.4 summarizes the numerical results of applying the proposed FPTAS to solve Instance D. Instance D is similar to Instance C but more general. Therefore, it can better evaluate the performance of the proposed FPTAS. The same observations as on Instance C can be made on Instance D. Therefore, we can conclude that the proposed FPTAS can solve the ISSP correctly and efficiently.

## 5 Concluding Remarks

In this paper, we considered the NP-hard ISSP, which is a generalization of the well-known SSP. We first showed that the ISSP can be equivalently reformulated as a 0-1 KP. This reformulation implies that the ISSP is easier to solve than the 0-1 KP, since any algorithms designed for the 0-1 KP can be used to solve the ISSP. Moreover, we identified several polynomial time solvable subclasses of the ISSP and thus clearly delineated a set of computationally tractable problems within the general class of NP-hard ISSPs. Then, by exploiting a new solution structure of the ISSP, we proposed a new FPTAS for it. Compared to the currently best known FPTAS, the proposed one has a comparable time complexity but a significantly lower space complexity. Numerical results demonstrate the correctness and efficiency of the proposed FPTAS.

The cardinality constrained ISSP [8] is an extension of the ISSP with an extra cardinality constraint  $\|x\|_0 \leq k_{\max}$ , where  $\|x\|_0$  denotes the number of nonzero elements in  $x$ . The proposed FPTAS for the ISSP in this paper can be modified to solve the cardinality constrained ISSP. The major modification is that  $k_{\max}$  of dynamic programming arrays  $\tilde{\Delta}_k$ ,  $k = 1, 2, \dots, k_{\max}$  need to be introduced in the proposed FPTAS for the ISSP, where  $\tilde{\Delta}_k$  is the same as the dynamic programming array in Algorithm 3.3 except that each element in  $\tilde{\Delta}_k$  is a summation of exactly

**Table 4.3** Numerical Results of Instance C

$n$	$c$	$\epsilon = 10\%$		$\epsilon = 1\%$		$\epsilon = 0.1\%$	
		relative error	time(s)	relative error	time(s)	relative error	time(s)
1,000	1.5	7.158% (9.218%)	0.000 (0.001)	0.076% (0.209%)	0.001 (0.001)	0.000% (0.000%)	0.006 (0.006)
	1.3	7.874% (9.415%)	0.000 (0.001)	0.308% (0.470%)	0.001 (0.001)	0.000% (0.000%)	0.006 (0.008)
	1.1	8.029% (9.771%)	0.000 (0.001)	0.643% (0.783%)	0.001 (0.001)	0.000% (0.000%)	0.005 (0.007)
5,000	1.5	7.498% (9.614%)	0.002 (0.003)	0.581% (0.648%)	0.002 (0.002)	0.000% (0.000%)	0.014 (0.018)
	1.3	8.187% (9.733%)	0.002 (0.002)	0.696% (0.750%)	0.002 (0.002)	0.000% (0.000%)	0.014 (0.016)
	1.1	8.033% (9.893%)	0.002 (0.003)	0.848% (0.901%)	0.002 (0.004)	0.000% (0.000%)	0.014 (0.015)
10,000	1.5	7.373% (9.732%)	0.004 (0.005)	0.665% (0.746%)	0.003 (0.004)	0.000% (0.000%)	0.020 (0.021)
	1.3	8.144% (9.831%)	0.004 (0.005)	0.755% (0.820%)	0.003 (0.004)	0.000% (0.000%)	0.020 (0.022)
	1.1	8.168% (9.921%)	0.004 (0.005)	0.892% (0.927%)	0.004 (0.005)	0.009% (0.029%)	0.020 (0.021)
50,000	1.5	7.304% (9.880%)	0.020 (0.027)	0.701% (0.883%)	0.017 (0.024)	0.000% (0.000%)	0.049 (0.053)
	1.3	7.954% (9.917%)	0.021 (0.027)	0.764% (0.918%)	0.018 (0.026)	0.008% (0.020%)	0.049 (0.051)
	1.1	7.667% (9.965%)	0.023 (0.026)	0.900% (0.967%)	0.019 (0.025)	0.053% (0.067%)	0.053 (0.062)
100,000	1.5	7.570% (9.911%)	0.037 (0.054)	0.749% (0.916%)	0.033 (0.049)	0.010% (0.019%)	0.073 (0.077)
	1.3	7.999% (9.941%)	0.042 (0.050)	0.795% (0.942%)	0.038 (0.050)	0.035% (0.043%)	0.074 (0.086)
	1.1	7.416% (9.974%)	0.045 (0.050)	0.914% (0.977%)	0.041 (0.055)	0.066% (0.077%)	0.084 (0.101)

**Table 4.4** Numerical Results of Instance D

$n$	$C$	$\epsilon = 10\%$		$\epsilon = 1\%$		$\epsilon = 0.1\%$	
		relative error	time(s)	relative error	time(s)	relative error	time(s)
1,000	1.5	5.353% (9.881%)	0.000 (0.001)	0.319% (0.945%)	0.000 (0.001)	0.003% (0.049%)	0.001 (0.003)
	1.3	4.830% (9.866%)	0.000 (0.000)	0.247% (0.926%)	0.000 (0.001)	0.004% (0.057%)	0.001 (0.003)
	1.1	4.891% (9.851%)	0.000 (0.001)	0.239% (0.931%)	0.001 (0.002)	0.006% (0.086%)	0.002 (0.005)
5,000	1.5	5.986% (9.979%)	0.001 (0.001)	0.392% (0.974%)	0.001 (0.003)	0.017% (0.082%)	0.002 (0.005)
	1.3	4.576% (9.805%)	0.001 (0.001)	0.362% (0.991%)	0.001 (0.003)	0.018% (0.087%)	0.002 (0.007)
	1.1	5.232% (9.949%)	0.001 (0.002)	0.314% (0.986%)	0.002 (0.004)	0.017% (0.091%)	0.004 (0.010)
10,000	1.5	6.375% (9.943%)	0.001 (0.002)	0.324% (0.985%)	0.002 (0.005)	0.023% (0.091%)	0.003 (0.009)
	1.3	6.511% (9.988%)	0.001 (0.002)	0.397% (0.979%)	0.002 (0.004)	0.019% (0.088%)	0.003 (0.011)
	1.1	5.191% (9.967%)	0.002 (0.004)	0.324% (0.982%)	0.003 (0.007)	0.017% (0.096%)	0.007 (0.014)
50,000	1.5	6.637% (9.969%)	0.005 (0.007)	0.415% (0.987%)	0.006 (0.010)	0.033% (0.098%)	0.010 (0.021)
	1.3	6.148% (9.982%)	0.005 (0.008)	0.372% (0.991%)	0.007 (0.014)	0.032% (0.099%)	0.012 (0.034)
	1.1	5.695% (9.991%)	0.006 (0.009)	0.376% (0.995%)	0.009 (0.019)	0.026% (0.095%)	0.021 (0.054)
100,000	1.5	6.477% (9.994%)	0.010 (0.013)	0.389% (0.990%)	0.012 (0.023)	0.030% (0.097%)	0.018 (0.046)
	1.3	6.256% (9.993%)	0.011 (0.014)	0.438% (0.999%)	0.013 (0.022)	0.029% (0.098%)	0.022 (0.055)
	1.1	6.245% (9.983%)	0.012 (0.017)	0.431% (0.980%)	0.017 (0.029)	0.025% (0.098%)	0.035 (0.096)

$k$  end points of the intervals  $\{[a_{i,1}, a_{i,2}]\}_{i=1}^n$ . In this way, the above modified algorithm is able to efficiently deal with the cardinality constraint. By using the same argument as in Theorems 3.10 and 3.11, it can be shown that the above modified algorithm is an FPTAS for the cardinality constrained ISSP and its time and space complexities is  $\mathcal{O}(n \max\{k_{\max}/\epsilon, \log n\})$  and  $\mathcal{O}(n + k_{\max}/\epsilon)$ , respectively.

## Appendix A: Proofs of Lemmas/Theorems/Corollaries

### Proof of Theorem 2.1

*Proof* We prove the theorem by dividing the proof into two cases, i.e., whether there exists binary  $\{\bar{y}_i\}_{i=1}^n$  such that

$$\sum_{i=1}^n a_{i,1} \bar{y}_i \leq T \leq \sum_{i=1}^n a_{i,2} \bar{y}_i. \quad (5.1)$$

Case A: there exists binary  $\{\bar{y}_i\}_{i=1}^n$  such that (5.1) holds true. Without loss of generality, assume

$$\bar{y}_i = 1, \quad i = 1, 2, \dots, \bar{I}; \quad \bar{y}_i = 0, \quad i = \bar{I} + 1, \dots, n. \quad (5.2)$$

In this case, we claim that both the optimal value of problems (2.1) and (2.2) are equal to  $T$ . Let us argue the above claim holds. First, it follows from (5.1) that  $\{\bar{y}_i\}_{i=1}^n$  is feasible to problem (2.2) and the optimal value of problem (2.2) is equal to  $T$ . Now, we evaluate the objective function of problem (2.1) at point  $\{\bar{y}_i\}_{i=1}^n$ :

$$g(z) := \sum_{i=1}^n (a_{i,1} \bar{y}_i + z_i) = \sum_{i=1}^{\bar{I}} a_{i,1} + \sum_{i=1}^{\bar{I}} z_i,$$

where the last equality is due to the assumption (5.2). Since  $z_i$  can take any value in the interval  $[0, a_{i,2} - a_{i,1}]$  for each  $i = 1, 2, \dots, \bar{I}$ , we know that  $g(z)$  can take any value between  $\left[\sum_{i=1}^{\bar{I}} a_{i,1}, \sum_{i=1}^{\bar{I}} a_{i,2}\right]$ . Combining this, (5.1), and the constraint  $\sum_{i=1}^n (a_{i,1} \bar{y}_i + z_i) \leq T$ , we know that the optimal value of problem (2.1) is exactly equal to  $T$ . As a matter of fact, an integer solution  $\{\bar{z}_i\}_{i=1}^n$  to achieve the optimal value  $T$  can be found as follows. Calculate

$$v^i = \sum_{l=1}^i a_{l,2} + \sum_{l=i+1}^{\bar{I}} a_{l,1}, \quad i = 0, 1, \dots, \bar{I} - 1.$$

Then there must exist an index  $i^* \in \{0, 1, \dots, \bar{I} - 1\}$  such that  $v^{i^*} < T \leq v^{i^*+1}$ , and  $\{\bar{z}_i\}_{i=1}^n$  to achieve the optimal value  $T$  is given by

$$\bar{z}_i = \begin{cases} a_{i,2} - a_{i,1}, & i = 1, \dots, i^*, \\ T - v^{i^*}, & i = i^* + 1, \\ 0, & i = i^* + 2, \dots, n. \end{cases} \quad (5.3)$$

We now show that there is a correspondence between the solutions of problems (2.1) and (2.2). On one hand, for any solution  $\{y_i^*, z_i^*\}_{i=1}^n$  of problem (2.1) achieving the optimal value  $T$  (from the above claim), we have  $\sum_{i=1}^n a_{i,1}y_i^* \leq \sum_{i=1}^n (a_{i,1}y_i^* + z_i^*) = T$ , and  $\sum_{i=1}^n a_{i,2}y_i^* \geq \sum_{i=1}^n (a_{i,1}y_i^* + z_i^*) = T$ . This immediately shows that  $\{y_i^*\}_{i=1}^n$  is a solution of problem (2.2). On the other hand, suppose that  $\{y_i^*\}_{i=1}^n$  is a solution of problem (2.2) achieving the optimal value  $T$  (from the above claim). Then, there must hold  $\sum_{i=1}^n a_{i,1}y_i^* \leq T \leq \sum_{i=1}^n a_{i,2}y_i^*$ . By using the same argument as in the proof of the above claim, we can show that there exists integers  $\{z_i^*\}_{i=1}^n$  such that  $\{y_i^*, z_i^*\}_{i=1}^n$  is a solution of problem (2.1).

Case B: there does not exist binary  $\{\bar{y}_i\}_{i=1}^n$  such that (5.1) holds true. This means that, for any binary  $\{y_i\}_{i=1}^n$  satisfying  $\sum_{i=1}^n a_{i,1}y_i \leq T$ , we must have  $\sum_{i=1}^n a_{i,2}y_i < T$ . Therefore, for any binary  $\{y_i\}_{i=1}^n$ , we have

$$\sum_{i=1}^n a_{i,1}y_i \leq T \iff \sum_{i=1}^n a_{i,2}y_i < T. \quad (5.4)$$

Hence, the optimal value of problem (2.2) is strictly less than  $T$  and it is equivalent to

$$\begin{aligned} & \max_y \sum_{i=1}^n a_{i,2}y_i \\ & \text{s.t. } \sum_{i=1}^n a_{i,1}y_i \leq T, \\ & y_i \in \{0, 1\}, \quad i = 1, 2, \dots, n. \end{aligned} \quad (5.5)$$

Moreover, the relation (5.4) implies that the solution of problem (2.1) must satisfy  $z_i = y_i(a_{i,2} - a_{i,1})$  for all  $i = 1, 2, \dots, n$ , since this maximizes the objective without violating the constraints. Combining this and (5.4), we know that problem (2.1) is equivalent to problem (5.5).

Combining Cases A and B, we can conclude that ISSP (2.1) is equivalent to problem (2.2). This completes the proof.  $\square$

## Proof of Theorem 2.2

*Proof* We prove the theorem by considering the following two cases.

Case A:  $T \geq \sum_{i=1}^n a_{i,1}$ . In this case, it is simple to find the solution of ISSP (2.1): if  $T \leq \sum_{i=1}^n a_{i,2}$ , then the solution to ISSP (2.1) is  $y_i = 1$  for all  $i = 1, 2, \dots, n$  and  $z_i$  is given by (5.3) with  $\bar{I}$  there being replaced with  $n$ ; otherwise the solution to ISSP (2.1) is  $y_i = 1$  and  $z_i = a_{i,2} - a_{i,1}$  for all  $i = 1, 2, \dots, n$ .

Case B:  $T < \sum_{i=1}^n a_{i,1}$ . Then, we can find  $I \geq 0$  such that

$$\sum_{i=1}^I a_{i,1} \leq T < \sum_{i=1}^{I+1} a_{i,1} \quad (5.6)$$

in polynomial time. If

$$\sum_{i=1}^I a_{i,2} > T, \quad (5.7)$$

then we can easily construct a solution such that the optimal value of ISSP (2.1) is  $T$  in polynomial time as in Case A of the proof of Theorem 2.1 and thus ISSP (2.1) is polynomial time solvable. Next, we show the truth of (5.7) under the assumption (2.4).

From the right hand side of (5.6), we get

$$T < \sum_{i=1}^{I+1} a_{i,1} \leq (I+1) \max_{1 \leq i \leq n} \{a_{i,1}\},$$

which further implies

$$I \geq \left\lfloor \frac{T}{\max_{1 \leq i \leq n} \{a_{i,1}\}} \right\rfloor.$$

Combining this with (2.4) yields

$$I \geq \left\lfloor \frac{\max_{1 \leq i \leq n} \{a_{i,1}\}}{\min_{1 \leq i \leq n} \{a_{i,2} - a_{i,1}\}} \right\rfloor, \quad (5.8)$$

which means

$$I \min_{1 \leq i \leq n} \{a_{i,2} - a_{i,1}\} \geq \max_{1 \leq i \leq n} \{a_{i,1}\}. \quad (5.9)$$

Now, we can use (5.6) and (5.9) to obtain (5.7). In particular, we have

$$\begin{aligned} \sum_{i=1}^I a_{i,2} &= \sum_{i=1}^I (a_{i,2} - a_{i,1}) + \sum_{i=1}^I a_{i,1} \\ &\geq I \min_{1 \leq i \leq n} \{a_{i,2} - a_{i,1}\} + \sum_{i=1}^I a_{i,1} \\ &\geq \max_{1 \leq i \leq n} \{a_{i,1}\} + \sum_{i=1}^I a_{i,1} \\ &> T, \end{aligned}$$

where the second inequality is due to (5.9) and the last inequality is due to the right hand side of (5.6). This completes the proof of Theorem 2.2.  $\square$

### Proof of Theorem 2.3

*Proof* Without loss of generality, assume  $a_{1,2} = \max_{1 \leq i \leq n} a_{i,2}$  in this proof. We have the following result.

**Lemma 5.1** *Suppose (2.5) holds true for some  $c > 1$  and  $T$  obeys the uniform distribution over the interval  $\left(a_{1,2}, \sum_{i=1}^n a_{i,2}\right]$ . Then, the probability that ISSP (1.3)*

is polynomial time solvable is at least

$$\frac{\sum_{j=2}^n \min \left\{ \sum_{i=1}^j \left(1 - \frac{1}{c}\right) a_{i,2}, a_{j,2} \right\}}{\sum_{j=2}^n a_{j,2}}.$$

Combining Lemma 5.1 and the following inequality

$$\begin{aligned} \sum_{j=2}^n \min \left\{ \sum_{i=1}^j \left(1 - \frac{1}{c}\right) a_{i,2}, a_{j,2} \right\} &\geq \sum_{j=2}^n \min \left\{ 2 \left(1 - \frac{1}{c}\right) a_{j,2}, a_{j,2} \right\}, \\ &= \min \left\{ 2 \left(1 - \frac{1}{c}\right), 1 \right\} \sum_{j=2}^n a_{j,2}, \end{aligned}$$

we immediately obtain Theorem 2.3.

Next, we show the truth of Lemma 5.1. The interval  $\left(a_{1,2}, \sum_{i=1}^n a_{i,2}\right]$  can be partitioned as follows:

$$\left(a_{1,2}, \sum_{i=1}^n a_{i,2}\right] = \bigcup_{j=2}^n \left(\sum_{i=1}^{j-1} a_{i,2}, \sum_{i=1}^j a_{i,2}\right]. \quad (5.10)$$

As shown in Case A of Theorem 2.1, for any  $T \in \left[\sum_{i=1}^j a_{i,1}, \sum_{i=1}^j a_{i,2}\right]$  with  $j \in \{1, 2, \dots, n\}$ , ISSP (1.3) is polynomial time solvable. Therefore, the probability that ISSP (1.3) is polynomial time solvable is greater than or equal to

$$\frac{\left| \bigcup_{j=2}^n \left( \left[ \sum_{i=1}^j a_{i,1}, \sum_{i=1}^j a_{i,2} \right] \cap \left( \sum_{i=1}^{j-1} a_{i,2}, \sum_{i=1}^j a_{i,2} \right] \right) \right|}{\left| \bigcup_{j=2}^n \left( \sum_{i=1}^{j-1} a_{i,2}, \sum_{i=1}^j a_{i,2} \right] \right|}, \quad (5.11)$$

where  $|\cdot|$  denotes the length of the corresponding set. Moreover, it can be verified that the denominator of (5.11) is equal to  $\sum_{j=2}^n a_{j,2}$  and the numerator of (5.11)

is lower bounded by  $\sum_{j=2}^n \min \left\{ \sum_{i=1}^j \left(1 - \frac{1}{c}\right) a_{i,2}, a_{j,2} \right\}$ . This completes the proof of Lemma 5.1.  $\square$

#### Proof of Lemma 3.4

*Proof* By Lemma 3.3, the ISSP has an optimal solution with at most one midrange element and all left anchored intervals precede the midrange interval and all right

anchored intervals follow the midrange interval. Without loss of generality, suppose  $\{x_i^*\}_{i=1}^n$  is such an optimal solution with  $[a_{j,1}, a_{j,2}]$  being the only midrange interval (i.e.,  $x_j^* \in (a_{j,1}, a_{j,2})$ ) and  $[a_{k,1}, a_{k,2}]$  with  $k > j$  being the last right anchored interval (i.e.,  $x_k^* = a_{k,2}$ ). Next, we construct an optimal solution  $\{\tilde{x}_i^*\}_{i=1}^n$  as follows:

$$\tilde{x}_i^* = \begin{cases} x_i^*, & \text{if } i \neq j \text{ or } k, \\ a_{k,2} - a_{j,2} + x_j^*, & \text{if } i = k, \\ a_{j,2}, & \text{if } i = j. \end{cases}$$

Since  $a_{j,2} - a_{j,1} \leq a_{k,2} - a_{k,1}$  and  $x_j^* \in (a_{j,1}, a_{j,2})$ , it follows that

$$a_{k,1} = a_{k,2} - (a_{k,2} - a_{k,1}) \leq a_{k,2} - (a_{j,2} - a_{j,1}) < a_{k,2} - (a_{j,2} - x_j^*) < a_{k,2}.$$

Therefore,  $\{\tilde{x}_i^*\}_{i=1}^n$  constructed in the above is feasible and optimal to the ISSP. Moreover, the solution  $\{\tilde{x}_i^*\}_{i=1}^n$  contains only one midrange element  $\tilde{x}_k^*$  and there are neither left nor right anchored intervals following this midrange interval. The proof is completed.  $\square$

### Proof of Lemma 3.5

*Proof* We prove the lemma by induction. Obviously, the lemma is true for  $i = 1$ . Assume it is true for some  $i \geq 1$ . Next, we show it is also true for  $i + 1$ . By the assumption and the fact  $\Delta_i^* \subseteq \Delta_{i+1}^*$ , we only need to consider the elements  $\delta$  in the set

$$\Delta_{i+1}^* \setminus \Delta_i^* = \{\delta + a_{i+1,1}, \delta + a_{i+1,2} \mid \delta \in \Delta_i^*\} \cup \{a_{i+1,1}, a_{i+1,2}\}. \quad (5.12)$$

The lemma is trivially true if  $\delta = a_{i+1,1}$  or  $\delta = a_{i+1,2}$ . It remains to show that the lemma is true for  $\delta = \delta' + v \leq \bar{T}$  where  $\delta' \in \Delta_i^*$  and  $v \in \{a_{i+1,1}, a_{i+1,2}\}$ . According to the assumption, we divide the subsequent proof into two Cases A and B.

Case A: there exist  $\underline{\delta}'$ ,  $\overline{\delta}' \in \Delta_i$  such that

$$\underline{\delta}' \leq \delta' \leq \overline{\delta}' \text{ and } \overline{\delta}' - \underline{\delta}' \leq \epsilon T. \quad (5.13)$$

In this case, we further consider the following three subcases A1, A2, and A3.

- A1:  $\underline{\delta}' + v \in I_j$  and  $\overline{\delta}' + v \in I_j$  for some  $j$ . Then, let  $\underline{\delta}$  and  $\overline{\delta}$  be the minimum and maximum values of  $\Delta_{i+1}$  in the subinterval  $I_j$ , respectively. It is simple to check that  $\underline{\delta} \leq \underline{\delta}' + v \leq \delta \leq \overline{\delta}' + v \leq \overline{\delta}$  and  $\overline{\delta} - \underline{\delta} \leq \epsilon T$ .
- A2:  $\underline{\delta}' + v \in I_j$  and  $\overline{\delta}' + v \in I_{j+1}$  for some  $j$ . Let  $\underline{\delta}_j$  and  $\overline{\delta}_j$  ( $\underline{\delta}_{j+1}$  and  $\overline{\delta}_{j+1}$ ) be the minimum and maximum values of  $\Delta_{i+1}$  in the subinterval  $I_j$  ( $I_{j+1}$ ), respectively. Then, by (5.13) and the assumption in the subcase A2, there must exist  $\underline{\delta}_j, \overline{\delta}_j, \underline{\delta}_{j+1}, \overline{\delta}_{j+1} \in \Delta_{i+1}$  such that  $\underline{\delta}_j \leq \underline{\delta}' + v \leq \overline{\delta}_j \leq \underline{\delta}_{j+1} \leq \overline{\delta}' + v \leq \overline{\delta}_{j+1}$ ,  $\overline{\delta}_j - \underline{\delta}_j \leq \epsilon T$ ,  $\underline{\delta}_{j+1} - \overline{\delta}_j \leq (\overline{\delta}' + v) - (\underline{\delta}' + v) \leq \epsilon T$ , and  $\overline{\delta}_{j+1} - \underline{\delta}_{j+1} \leq \epsilon T$ . If  $\delta' + v \in [\underline{\delta}_j, \overline{\delta}_j]$ , let  $\underline{\delta} = \underline{\delta}_j$  and  $\overline{\delta} = \overline{\delta}_j$ ; if  $\delta' + v \in [\overline{\delta}_j, \underline{\delta}_{j+1}]$ , let  $\underline{\delta} = \overline{\delta}_j$  and  $\overline{\delta} = \underline{\delta}_{j+1}$ ; and if  $\delta' + v \in [\underline{\delta}_{j+1}, \overline{\delta}_{j+1}]$ , let  $\underline{\delta} = \underline{\delta}_{j+1}$  and  $\overline{\delta} = \overline{\delta}_{j+1}$ . It is simple to verify that  $\underline{\delta}$  and  $\overline{\delta}$  constructed in the above satisfy  $\underline{\delta} \leq \delta \leq \overline{\delta}$  and  $\overline{\delta} - \underline{\delta} \leq \epsilon T$ .



A3:  $\underline{\delta}' + v \in I_j$  for some  $j$  and  $\bar{\delta}' + v > \tilde{T}$ . Since  $\tilde{T} - \epsilon T$  and  $\tilde{T}$  are in different subintervals, we assume  $\tilde{T} - \epsilon T \in I_j$  and  $\tilde{T} \in I_{j+1}$  without loss of generality. Then, we get either  $\delta' + v \in I_j$  or  $\delta' + v \in I_{j+1}$ . If  $\delta' + v \in I_j$ , let  $\underline{\delta}$  and  $\bar{\delta}(\leq \tilde{T})$  be the minimum and maximum values of  $\Delta_{i+1}$  in the subinterval  $I_j$ , respectively, and we thus have  $\underline{\delta} \leq \delta \leq \bar{\delta}$  and  $\bar{\delta} - \underline{\delta} \leq \epsilon T$ . If  $\delta' + v \in I_{j+1}$ , let  $\underline{\delta}$  be the minimum value of  $\Delta_{i+1}$  in  $I_{j+1}$ . Since  $\tilde{T} - \epsilon T \in I_j$ , it follows that  $\tilde{T} - \epsilon T \leq \underline{\delta} \leq \delta \leq \tilde{T}$ .

Case B: there exists  $\underline{\delta}' \in \Delta_i$  such that  $\tilde{T} - \epsilon T \leq \underline{\delta}' \leq \delta' \leq \tilde{T}$ . Without loss of generality, assume  $\tilde{T} - \epsilon T \in I_j$  and  $\tilde{T} \in I_{j+1}$  for some  $j$ . We can show that the lemma is true for this case by using the same argument as in the above subcase A3. More specifically, we can show that: if  $\delta \in I_j$ , there exist  $\underline{\delta}, \bar{\delta} \in \Delta_{i+1}$  such that  $\underline{\delta} \leq \delta \leq \bar{\delta}$  and  $\bar{\delta} - \underline{\delta} \leq \epsilon T$ ; and if  $\delta \in I_{j+1}$ , there exists  $\underline{\delta}$  such that  $\tilde{T} - \epsilon T \leq \underline{\delta} \leq \delta \leq \tilde{T}$ .

This completes the proof of Lemma 3.5.  $\square$

### Proof of Corollary 3.6

*Proof* For  $\delta^* \in \Delta_{\tilde{A}}^*$ , by Lemma 3.5, we have one of the following two statements:

1. there exist  $\underline{\delta}, \bar{\delta} \in \Delta_{\tilde{A}}$  such that  $\underline{\delta} \leq \delta^* \leq \bar{\delta}$  and  $\bar{\delta} - \underline{\delta} \leq \epsilon T$ ;
2. there exists  $\underline{\delta} \in \Delta_{\tilde{A}}$  such that  $\tilde{T} - \epsilon T \leq \underline{\delta} \leq \delta^* \leq \tilde{T}$ .

If the first statement is true, let  $\delta$  be  $\bar{\delta}$  there. Since  $\delta^*$  is the optimal value of the ISSP, we must have  $\delta = \delta^*$ . If the second statement is true, let  $\delta$  be  $\underline{\delta}$  there. Then, we immediately obtain the desired result.  $\square$

### Proof of Lemma 3.7

*Proof* Since  $\delta \in \Delta_{\tilde{A}} \subseteq \Delta_{\tilde{A}}^*$ , it follows that there must exist  $\delta_1 \in \{0\} \cup \Delta_{\tilde{A}_1}^*$  and  $\delta_2 \in \{0\} \cup \Delta_{\tilde{A}_2}^*$  such that  $\delta_1 + \delta_2 = \delta$ . Invoking Lemma 3.5 again, we have one of the following two statements:

1. there exists  $\underline{\delta}_1 \in \Delta_{\tilde{A}_1}$  such that  $\tilde{T} - \epsilon T \leq \underline{\delta}_1 \leq \delta_1 \leq \tilde{T}$  or there exists  $\underline{\delta}_2 \in \Delta_{\tilde{A}_2}$  such that  $\tilde{T} - \epsilon T \leq \underline{\delta}_2 \leq \delta_2 \leq \tilde{T}$ ;
2. there exist  $\underline{\delta}_1 \in \Delta_{\tilde{A}_1}, \bar{\delta}_1 \in \Delta_{\tilde{A}_1}, \underline{\delta}_2 \in \Delta_{\tilde{A}_2}, \bar{\delta}_2 \in \Delta_{\tilde{A}_2}$  such that  $\underline{\delta}_1 \leq \delta_1 \leq \bar{\delta}_1$ ,  $\underline{\delta}_2 \leq \delta_2 \leq \bar{\delta}_2$ ,  $0 \leq \bar{\delta}_1 - \underline{\delta}_1 \leq \epsilon T$ , and  $0 \leq \bar{\delta}_2 - \underline{\delta}_2 \leq \epsilon T$ .

If the first statement is true, then let  $(u_1, u_2) = (\underline{\delta}_1, 0)$  or  $(u_1, u_2) = (0, \underline{\delta}_2)$ . Obviously,  $u_1$  and  $u_2$  defined in the above satisfy  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ . It remains to show the lemma if the second statement is true. We consider the following three cases separately.

Case A:  $\underline{\delta}_1 + \underline{\delta}_2 \geq \tilde{T} - \epsilon T$ . In this case, let  $(u_1, u_2) = (\underline{\delta}_1, \underline{\delta}_2)$ . Then, combining the facts  $\delta_1 + \delta_2 = \delta$ ,  $\underline{\delta}_1 \leq \delta_1$ ,  $\underline{\delta}_2 \leq \delta_2$  and  $\delta \leq \tilde{T}$ , we immediately obtain  $\tilde{T} - \epsilon T \leq u_1 + u_2 = \underline{\delta}_1 + \underline{\delta}_2 \leq \delta_1 + \delta_2 = \delta \leq \tilde{T}$ .

Case B:  $\bar{\delta}_1 + \bar{\delta}_2 \leq \tilde{T}$ . In this case, let  $(u_1, u_2) = (\bar{\delta}_1, \bar{\delta}_2)$ . We can use essentially the same argument as in the above Case A to show  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ .

Case C:  $\underline{\delta}_1 + \underline{\delta}_2 < \tilde{T} - \epsilon T$  and  $\bar{\delta}_1 + \bar{\delta}_2 > \tilde{T}$ . In this case, let  $(u_1, u_2) = (\bar{\delta}_1, \underline{\delta}_2)$ . Combining the conditions assumed in this case,  $\bar{\delta}_1 - \underline{\delta}_1 \leq \epsilon T$ , and  $\bar{\delta}_2 - \underline{\delta}_2 \leq \epsilon T$ ,

we immediately obtain  $u_1 + u_2 = \overline{\delta_1} + \underline{\delta_2} = (\overline{\delta_1} + \overline{\delta_2}) - (\overline{\delta_2} - \underline{\delta_2}) > \tilde{T} - \epsilon T$  and  $u_1 + u_2 = \overline{\delta_1} + \underline{\delta_2} = (\underline{\delta_1} + \underline{\delta_2}) + (\overline{\delta_1} - \underline{\delta_1}) < \tilde{T}$ .

From the above analysis, we conclude that there exist  $u_1 \in \{0\} \cup \Delta_{\tilde{A}_1}$  and  $u_2 \in \{0\} \cup \Delta_{\tilde{A}_2}$  such that  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ . The proof is completed.  $\square$

### Proof of Lemma 3.8

*Proof* By the assumption, the largest number in  $\Delta_{\tilde{A}}^*$  associated with the target  $\tilde{T}$  must be in the interval  $[\tilde{T} - \epsilon T, \tilde{T}]$ . Then, it follows from Corollary 3.6 that the largest number in  $\Delta_{\tilde{A}}$  associated with the target  $\tilde{T}$  must also lie in the interval  $[\tilde{T} - \epsilon T, \tilde{T}]$ . Recall the procedure **backtracking**, we know the following facts: when the procedure starts, the largest number in  $\Delta_{\tilde{A}}$  associated with the target  $\tilde{T}$  (i.e.,  $u = \max\{\delta^+(j), \delta^-(j) \mid \delta^+(j) \leq \tilde{T}, \delta^-(j) \leq \tilde{T}\}$ ) is found in line 1; the recent intervals which contribute to generate  $u$  are backtracked in lines 4 – 18; and when the procedure terminates,  $y + u$  is in the interval  $[\tilde{T} - \epsilon T, \tilde{T}]$ . Since  $u$  is generated by the procedure **relaxed dynamic programming**, it follows that  $u \in \{0\} \cup \Delta_{\tilde{A} \setminus \Lambda^E}$ . This further implies that there exists  $\delta \in \{0\} \cup \Delta_{\tilde{A} \setminus \Lambda^E}^*$  such that  $\tilde{T} - \epsilon T \leq y + \delta \leq \tilde{T}$ . This completes the proof of Lemma 3.8.  $\square$

### Proof of Lemma 3.9

*Proof* First, as argued in the proof of Lemma 3.8, we know that the largest number in  $\Delta_{\tilde{A}}$  associated with the target  $\tilde{T}$  must also lie in the interval  $[\tilde{T} - \epsilon T, \tilde{T}]$ . By Lemma 3.7, we can split  $\tilde{A}$  into  $\tilde{A}_1$  and  $\tilde{A}_2$  as in lines 2 and 3 of the procedure **divide and conquer**, and find  $u_1 \in \{0\} \cup \Delta_{\tilde{A}_1}$  and  $u_2 \in \{0\} \cup \Delta_{\tilde{A}_2}$  such that  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$ . Without loss of generality, assume both  $u_1$  and  $u_2$  are positive. Otherwise, if  $u_1 = 0$  ( $u_2 = 0$ ), then we can remove the intervals in  $\tilde{A}_1$  ( $\tilde{A}_2$ ) and split  $\tilde{A}_2$  ( $\tilde{A}_1$ ) again. This implies that there exists positive  $u_1 \in \Delta_{\tilde{A}_1}^*$  satisfying  $u_1 \in [\tilde{T} - u_2 - \epsilon T, \tilde{T} - u_2]$ . Moreover, by Lemma 3.8, we know that there exists  $\delta \in \{0\} \cup \Delta_{\tilde{A}_1 \setminus \Lambda^E}^*$  such that  $\delta \in [\tilde{T} - u_2 - y_1^B - \epsilon T, \tilde{T} - u_2 - y_1^B]$ , where  $y_1^B \geq 1$  is the output of line 7 of the procedure **divide and conquer**. This in turn shows that the assumption of Lemma 3.9 is satisfied for the procedure **divide and conquer** in line 10. Since at least one interval is removed after each recursive call, the recursive calls of the procedure **divide and conquer** will eventually end. Consequently, we get  $y_1^{DC} \in [\tilde{T} - u_2 - y_1^B - \epsilon T, \tilde{T} - u_2 - y_1^B]$ , which is equivalent to  $u_2 \in [\tilde{T} - y_1^B - y_1^{DC} - \epsilon T, \tilde{T} - y_1^B - y_1^{DC}]$ . Similar analysis applies to lines 13, 14, and 17 of the procedure **divide and conquer**. This completes the proof of Lemma 3.9.  $\square$

### Proof of Theorem 3.10

*Proof* By enumerating all intervals  $\{[a_{i,1}, a_{i,2}]\}_{i=1}^n$ , Algorithm 3.3 can successfully find the (possible) midrange interval  $[a_{m,1}, a_{m,2}]$  and the largest number  $\hat{\delta}$  in  $\Delta_{\Lambda \setminus \Lambda^E}$  associated with the target  $T - a_{m,1}$ . Suppose that  $\delta^*$  is the largest number

in  $\Delta_{\Lambda \setminus \Lambda^E}^*$  associated with the target  $T - a_{m,1}$ . Then, by Lemma 3.5, we have either  $\hat{\delta} = \delta^*$  or  $T - a_{m,1} - \epsilon T \leq \hat{\delta} \leq \delta^* \leq T - a_{m,1}$ . We consider the following four cases.

Case A:  $\hat{\delta} + \epsilon T < T - a_{m,1}$  and  $\hat{\delta} = \delta^*$ . In this case, we have  $\hat{\delta} \leq \delta^* \leq \hat{\delta} + \epsilon T$ . Using Lemma 3.9, we immediately obtain  $\hat{\delta} \leq \hat{T}^A \leq \hat{\delta} + \epsilon T$ , where  $\hat{T}^A$  is the output of the procedure **divide and conquer**  $(\Lambda \setminus \Lambda^E, \hat{\delta} + \epsilon T)$  in line 27 of Algorithm 3.3. Since  $\hat{\delta} = \delta^*$  is the largest number in  $\Delta_{\Lambda \setminus \Lambda^E}^*$  associated with the target  $T - a_{m,1}$ , it follows that  $\hat{T}^A = \hat{\delta} = \delta^*$ . Hence,

$$T^A = \hat{T}^A + \min \{a_{m,2}, T - \hat{T}^A\} = \delta^* + \min \{a_{m,2}, T - \delta^*\}$$

is the optimal value of the ISSP.

Case B:  $\hat{\delta} + \epsilon T < T - a_{m,1}$  and  $T - a_{m,1} - \epsilon T \leq \hat{\delta} \leq \delta^* \leq T - a_{m,1}$ . This case will not happen, since the two conditions contradict with each other.

Case C:  $\hat{\delta} + \epsilon T \geq T - a_{m,1}$  and  $\hat{\delta} = \delta^*$ . From these two conditions and the fact that  $\delta^*$  is the largest number in  $\Delta_{\Lambda \setminus \Lambda^E}^*$  associated with the target  $T - a_{m,1}$ , we obtain  $\delta^* \in [T - a_{m,1} - \epsilon T, T - a_{m,1}]$ . Then, by Lemma 3.9, we know that the returned approximate value of  $\hat{T}^A$  satisfies  $T - a_{m,1} \geq \hat{T}^A \geq T - a_{m,1} - \epsilon T$ , and  $T^A = \hat{T}^A + \min \{a_{m,2}, T - \hat{T}^A\} \geq \hat{T}^A + a_{m,1} \geq T - \epsilon T$ .

Case D:  $\hat{\delta} + \epsilon T \geq T - a_{m,1}$  and  $T - a_{m,1} - \epsilon T \leq \hat{\delta} \leq \delta^* \leq T - a_{m,1}$ . The same argument as in the above Case C shows that  $T - a_{m,1} \geq \hat{T}^A \geq T - a_{m,1} - \epsilon T$  and  $T^A \geq T - \epsilon T$ .

From the above analysis, we can conclude that Algorithm 3.3 either returns an optimal solution, or an approximate solution with the objective value being great than or equal to  $T - \epsilon T$ . The proof is completed.  $\square$

#### Proof of Theorem 3.11

*Proof* We analyze the time and space complexities of Algorithm 3.3 separately. We first consider the time complexity of Algorithm 3.3. The time complexity of sorting  $n$  intervals by length (line 1 of Algorithm 3.3) is  $\mathcal{O}(n \log n)$ . The procedure **relaxed dynamic programming** is called many times in Algorithm 3.3 and performing the procedure **relaxed dynamic programming** is the dominated computational cost in the recursive framework of the procedure **divide and conquer**. It is simple to see that the time complexity of performing the procedure **relaxed dynamic programming** with inputs  $(\tilde{\Lambda}, \tilde{T})$  is  $\mathcal{O}(\tilde{n}\tilde{l})$ , where  $\tilde{n} = |\tilde{\Lambda}|$  and  $\tilde{l} = \left\lceil \frac{\tilde{T}}{\epsilon \tilde{T}} \right\rceil$ .

Now, we bound the times that the procedure **divide and conquer** is performed. To do so, let us denote the root node of the recursive tree as level 0. Then, there are at most  $2^l \leq n$  nodes in the  $l$ -th level of the recursive tree. For ease of presentation, we assume that there are  $2^l$  nodes in the  $l$ -th level of the recursive tree and denote the targets of these  $2^l$  nodes by  $\tilde{T}_{l,1}, \tilde{T}_{l,2}, \dots, \tilde{T}_{l,2^l}$  and item sets of these  $2^l$  nodes by  $\tilde{\Lambda}_{l,1}, \tilde{\Lambda}_{l,2}, \dots, \tilde{\Lambda}_{l,2^l}$  for all  $l = 1, 2, \dots, \lceil \log n \rceil$ . Then, we must have  $\sum_{i=1}^{2^l} \tilde{T}_{l,i} \leq T$  for all  $l = 1, 2, \dots, \lceil \log n \rceil$  and  $|\tilde{\Lambda}_{l,i}| = \mathcal{O}(\frac{n}{2^l})$  for all

$i = 1, 2, \dots, 2^l$  and  $l = 1, 2, \dots, \lceil \log n \rceil$ . Therefore, the total time complexity of calling the procedure **divide and conquer** in Algorithm 3.3 is

$$\sum_{l=0}^{\lceil \log n \rceil} \sum_{i=1}^{2^l} \mathcal{O} \left( \left\lceil \frac{\tilde{T}_{l,i}}{\epsilon T} \right\rceil |\Lambda_{l,i}| \right) \leq \mathcal{O}(n/\epsilon),$$

and the total time complexity of Algorithm 3.3 is  $\mathcal{O}(n \max \{1/\epsilon, \log n\})$ .

Next, we consider the space complexity of Algorithm 3.3. It takes  $\mathcal{O}(n)$  space complexity to store the interval set  $\{[a_{i,1}, a_{i,2}]\}_{i=1}^n$ . The space complexity required to store the relaxed dynamic programming arrays  $\delta_1^-(\cdot), \delta_1^+(\cdot), \delta_2^-(\cdot), \delta_2^+(\cdot), d_{1,1}(\cdot), d_{1,2}(\cdot), d_{2,1}(\cdot), d_{2,2}(\cdot)$  is  $\mathcal{O}(1/\epsilon)$ . Since the memory space can be reused in the recursive calls of the procedure **divide and conquer**, we conclude that the total space complexity of Algorithm 3.3 is  $\mathcal{O}(n + 1/\epsilon)$ .  $\square$

## Appendix B: An Illustration of Algorithm 3.2 and Algorithm 3.3

To make Algorithm 3.2 and Algorithm 3.3 clear, an illustration of applying them to solve the following ISSP instance is given:

$$T = 100, \quad n = 4,$$

$$[a_{1,1}, a_{1,2}] = [10, 20], \quad [a_{2,1}, a_{2,2}] = [10, 25],$$

$$[a_{3,1}, a_{3,2}] = [60, 85], \quad [a_{4,1}, a_{4,2}] = [20, 50].$$

If Algorithm 3.2 is applied to solve the above instance: when  $i = 1$ , executing lines 4–12 gives

$$\delta^* = 0, \quad T^* = 20, \quad m = 1, \quad \Delta_1^* = \{10, 20\};$$

then  $i = 2$  and executing lines 4–12 gives

$$\delta^* = 20, \quad T^* = 45, \quad m = 2, \quad \Delta_2^* = \{10, 20, 25, 30, 35, 45\};$$

then  $i = 3$  and executing lines 4–12 gives  $\delta^* = 35, T^* = 100$ , and  $m = 3$ . Since  $T^* = T = 100$  when  $i = 3$ , Algorithm 3.2 goes to line 14 directly without computing

$$\Delta_3^* = \{10, 20, 25, 30, 35, 45, 60, 70, 80, 85, 90, 95\}.$$

Then, executing lines 14–17 gives  $\delta^* = 35$  and returns the optimal solution

$$x_1^* = 10, \quad x_2^* = 25, \quad x_3^* = 65, \quad x_4^* = 0.$$

It can be seen that  $[a_{1,1}, a_{1,2}]$  is a left anchored interval,  $[a_{2,1}, a_{2,2}]$  is a right anchored interval,  $[a_{3,1}, a_{3,2}]$  is the only midrange interval, and there is no left/right anchored intervals following the midrange interval. Therefore, the returned solution by Algorithm 3.2 satisfies the property in Lemma 3.4.

If Algorithm 3.3 with  $\epsilon = 0.2$  (and thus  $l = 5$ ) is applied to solve the above instance: when  $i = 1$ , executing lines 5–23 gives

$$\bar{\delta} = 0, \quad \hat{T} = 20, \quad \hat{\delta} = 0, \quad m = 1, \quad \tilde{\Delta} = \{10, 20\}, \quad \delta^-(1) = 10, \quad \delta^+(1) = 20,$$

and

$$\delta^-(2) = \delta^+(2) = \delta^-(3) = \delta^+(3) = \delta^-(4) = \delta^+(4) = \delta^-(5) = \delta^+(5) = 0;$$

then  $i = 2$  and executing lines 5–23 gives

$$\bar{\delta} = 20, \hat{T} = 45, \hat{\delta} = 20, m = 2, \tilde{\Delta} = \{10, 20, 25, 30, 35, 45\}, \delta^-(1) = 10, \delta^+(1) = 20,$$

and

$$\delta^-(2) = 25, \delta^+(2) = 35, \delta^-(3) = \delta^+(3) = 45, \delta^-(4) = \delta^+(4) = \delta^-(5) = \delta^+(5) = 0;$$

then  $i = 3$  and executing lines 5–23 gives

$$\bar{\delta} = 35, \hat{T} = 100, \hat{\delta} = 35, m = 3.$$

Since  $T^* = \hat{T} = 100$  when  $i = 3$ , Algorithm 3.3 goes to line 25 directly without computing

$$\tilde{\Delta} = \{60, 70, 80, 85, 95\}, \delta^-(1) = 10, \delta^+(1) = 20, \delta^-(2) = 25, \delta^+(2) = 35,$$

$$\delta^-(3) = 45, \delta^+(3) = 60, \delta^-(4) = 70, \delta^+(4) = 80, \delta^-(5) = 85, \delta^+(5) = 95.$$

Then, executing line 25 gives  $\Lambda^E = \{[60, 85], [20, 50]\}$  and Algorithm 3.3 calls the procedure **divide and conquer**  $\left(\Lambda \setminus \Lambda^E, \min \left\{ \hat{\delta} + \epsilon T, T - a_{m,1} \right\}\right)$ , where

$$\Lambda \setminus \Lambda^E = \{[10, 20], [10, 25]\}$$

and

$$\min \left\{ \hat{\delta} + \epsilon T, T - a_{m,1} \right\} = \min \{35 + 0.2 * 100, 100 - 60\} = 40.$$

With these inputs, line 1 of the procedure **divide and conquer** gives  $\tilde{\Lambda}_1 = \{[10, 20]\}$  and  $\tilde{\Lambda}_2 = \{[10, 25]\}$ ; line 2 of the procedure **divide and conquer** gives

$$\delta_1^-(1) = 10, \delta_1^+(1) = 20, \delta_1^-(2) = \delta_1^+(2) = 0,$$

$$d_{1,1}(\delta_1^-(1)) = d_{1,2}(\delta_1^-(1)) = d_{1,1}(\delta_1^+(1)) = 1, d_{1,2}(\delta_1^+(1)) = 2;$$

line 3 of the procedure **divide and conquer** gives

$$\delta_2^-(1) = \delta_2^+(1) = 10, \delta_2^-(2) = \delta_2^+(2) = 25,$$

$$d_{2,1}(\delta_2^-(1)) = d_{2,1}(\delta_2^+(1)) = 2, d_{2,2}(\delta_2^-(1)) = d_{2,2}(\delta_2^+(1)) = 1,$$

$$d_{2,1}(\delta_2^-(2)) = d_{2,2}(\delta_2^-(2)) = d_{2,1}(\delta_2^+(2)) = d_{2,2}(\delta_2^+(2)) = 2;$$

line 4 of the procedure **divide and conquer** finds  $(u_1, u_2) = (0, 25)$  satisfying  $u_1 \in \{0, 10, 20\}$ ,  $u_2 \in \{0, 10, 25\}$ , and  $20 = \tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T} = 40$ . Since both  $\tilde{\Lambda}_1$  and  $\tilde{\Lambda}_2$  contain only one interval, the procedure **backtracking** can successfully return the approximate solution. More specifically, the procedure **divide and conquer** skips lines 7 and 10; executes line 14 and gives  $y_2^B = 25$  and  $\Lambda^E = \{[a_{2,1}, a_{2,2}]\}$ ; and skips line 17. Then, line 27 of Algorithm 3.3 returns  $x_1^A = 0$ ,  $x_2^A = 25$ , and  $\hat{T}^A = 25$ ; line 28 of Algorithm 3.3 returns  $x_3^A = 75$ ; line 29 of Algorithm 3.3 returns  $x_4^A = 0$ ; and line 30 of Algorithm 3.3 returns  $T^A = 100$ . Again, the returned  $(1 - \epsilon)$ -optimal solution (actually the optimal

solution) by Algorithm 3.3 satisfies the property in Lemma 3.4, i.e.,  $[a_{2,1}, a_{2,2}]$  is a right anchored interval,  $[a_{3,1}, a_{3,2}]$  is the only midrange interval, and there is no left/right anchored intervals following the midrange interval.

Two remarks on Algorithm 3.2 and Algorithm 3.3 are in order.

First, although Algorithm 3.3 finds an optimal solution for the above ISSP instance, it cannot be guaranteed to do so for the general ISSP. The reason is that Algorithm 3.3 partitions the interval  $(0, T]$  into  $\lceil 1/\epsilon \rceil$  subintervals and stores only the smallest and largest values lying in the subintervals at each iteration. This is sharply different from Algorithm 3.2, where all values in  $\Delta_i^*$  are stored. For the above instance, when  $i = 2$ , we have

$$\left\{ \delta^-(k), \delta^+(k) \right\}_{k=1}^5 = \{10, 20, 25, 35, 45\} \subset \Delta_2^*;$$

and when  $i = 3$ , we have

$$\left\{ \delta^-(k), \delta^+(k) \right\}_{k=1}^5 = \{10, 20, 25, 35, 45, 60, 70, 80, 85, 95\} \subset \Delta_3^*.$$

Second, as mentioned below Lemma 3.7, the pair  $(u_1, u_2)$  satisfying the inequality  $\tilde{T} - \epsilon T \leq u_1 + u_2 \leq \tilde{T}$  is generally not unique and different choices of the pair  $(u_1, u_2)$  might lead to different approximate solutions. For example,  $(u_1, u_2)$  in the above instance can also be  $(10, 10)$ , which results in the approximate solution

$$x_1^A = 10, x_2^A = 10, x_3^A = 80, x_4^A = 0;$$

or can also be  $(10, 25)$ , which results in the approximate solution

$$x_1^A = 10, x_2^A = 25, x_3^A = 65, x_4^A = 0;$$

or can also be  $(20, 0)$ , which results in the approximate solution

$$x_1^A = 20, x_2^A = 0, x_3^A = 80, x_4^A = 0.$$

**Acknowledgements** We thank Prof. Nelson Maculan and Prof. Sergiy Butenko for the useful discussions on this paper.

## References

1. Brickell, E.F.: Solving low density knapsacks. In: *Advances in Cryptology*, pp. 25–37. Springer (1984)
2. Carrión, M., Arroyo, J.M.: A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems* **21**(3), 1371–1378 (2006)
3. Chvátal, V.: Hard knapsack problems. *Operations Research* **28**(6), 1402–1411 (1980)
4. Coster, M.J., Joux, A., LaMacchia, B.A., Odlyzko, A.M., Schnorr, C.P., Stern, J.: Improved low-density subset sum algorithms. *Computational Complexity* **2**(2), 111–128 (1992)
5. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM* **22**(4), 463–468 (1975)
6. Kellerer, H., Mansini, R., Pferschy, U., Speranza, M.G.: An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Sciences* **66**(2), 349–370 (2003)
7. Kellerer, H., Pferschy, U.: A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization* **3**(1), 59–71 (1999)

8. Kothari, A., Suri, S., Zhou, Y.H.: Interval subset sum and uniform-price auction clearing. In: *Computing and Combinatorics*, pp. 608–620. Springer (2005)
9. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. *Journal of the ACM* **32**(1), 229–246 (1985)
10. Lawler, E.L.: Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research* **4**(4), 339–356 (1979)
11. Magazine, M.J., Oguz, O.: A fully polynomial approximation algorithm for the 0–1 knapsack problem. *European Journal of Operational Research* **8**(3), 270–273 (1981)
12. Michael, R.G., Johnson, D.S.: *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman & Co., San Francisco (1979)
13. Pisinger, D.: Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms* **33**(1), 1–14 (1999)
14. Sun, X.L., Zheng, X.J., Li, D.: Recent advances in mathematical programming with semi-continuous variables and cardinality constraint. *Journal of the Operations Research Society of China* **1**(1), 55–77 (2013)